Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The CWI is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

CWI Tract 43

Distributed computing: structure and complexity

H.L. Bodlaender

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

1980 Mathematics Subject Classification: 68B20, 68C05.
1983 CR Classification Scheme: C.2.1.
ISBN 90 6196 327 3
NUGI-code: 811

ACKNOWLEDGEMENTS

A large number of people have helped me in many ways during the preparation of this work.

I am very grateful to Prof. J. van Leeuwen, for the many helpful suggestions and for the many hours he has spent carefully reading and improving my manuscripts.

I want to thank my colleagues at the department of Computer Science at the University of Utrecht, and especially Ingrid Birkhoff, Anneke Schoone, Gerard Tel and Harry Wijshoff for the pleasant environment and many stimulating discussions, which made my time at the department so agreeable.

Thanks are also due to Tineke Buys, Corine de Gee and Wanda Verwey for the enormous amounts of typing they have done.

I also want to thank P.W.H. Lemmens for pointing out several errors in the original version of section 5.6.2., for providing the example of table E (p. 156), and for some helpful comments on section 6.4. Prof. L.D. Wittie suggested to look at emulations of networks with buses (section 6.5.).

This work was supported by the Foundation for Computer Science (SION) of the Netherlands Organization for the Advancement of Pure Research, which I gratefully acknowledge here.

Finally I want to thank my family and friends and very especially my wife Brigit Bodlaender-Peters, for their encouragement, support, friendship and love, that I have experienced during the past years.

CONTENTS

CHAPTER ONE

INTRODUCTION

1. Historical notes. Already during the design of the first programmable computers at the late 1940's, it was noted that computations could be sped up by carrying out parts of it in parallel. However, to avoid a large number of difficult problems, intrinsic to parallel computation, von Neumann and his contemporaries proposed a strictly sequential computer model, in which one instruction is processed at a time. The sequential model of computation has dominated programming and the study of algorithms for many years.

Designers began to depart from the strictly sequential model of computing in the course of the 1950's, when it was observed that the performance of a system could benefit from a distribution of tasks. For example, letting the fast central processing unit (CPU) wait for the much slower input and output devices, results in the CPU being unnecessarily idle for long periods. By introducing separate processors for the I/O-devices and (thus) relieving the CPU from the task of I/O-control the delays could be avoided. Typically, the I/O-processors and the CPU work "concurrently", communicating only in a limited sense by signals. In modern multiprogrammed systems many separate jobs can be loaded and run simultaneously, with the system relying on the services of several logically "distributed" but communicating processes to deal with the stream of jobs.

The introduction of terminals made it possible to have access to a computer system from locations, remote from the central site. Hence, it can be seen as the first "geographical" distribution of a computer system. By not only connecting terminals to host computers, but also connecting host computers with each other (thus providing facilities like using special facilities at the other host, sending data to or accessing data at the other host, etc.), computer networks are created. Gradually, the computer networks that were designed and built became (and become) increasingly more complex and provided more and more complex facilities to the users.

Breaking up programs and tasks into processes (natural units of work) was another step towards (fully) distributed processing. By using more processing elements (in one machine), different processes can be carried out simultaneously on different processing elements, thus gaining on speed and (perhaps) reliability. (When one processing element goes down, other elements may take over its work.)

In recent years, the interest in distributed computer systems and distributed computing has increased very rapidly. We mention three of the most important reasons for this growing interest:

1. Sequential machines have approached the fundamental limits in the computing power they can provide. Essential improvements in the speed of these machines seems to be possible only if parallelization of the hardware and of the programs is used. The idea is that it is sometimes cheaper to use many relatively small and cheap components that work in parallel than one relatively large and expensive component. The approach leads to a distribution of logical units of works and of a distribution of control, that needs to be understood and managed. The rapid decrease of the size and cost of computer components makes it possible to design and build systems with large numbers of processors.

2. In a modern concurrent operating system many processes work concurrently, communicating with each other in some (limited) way. It seems natural to use a duplication of the hardware components in the system, like processing elements (some of which may be dedicated to specialized tasks), and to assign different processes to different processing elements. In this way we do not only gain on the speed of the computer system, but we also can obtain a more reliable computer system: when some (processing) elements go down or malfunction and others do not, the system as a whole may continue to work properly by letting the "good" components also do the work of the "bad" ones.

3. For various reasons one may want to use facilities of or provide facilities to computers that are on different locations and transfer data between computers that are on different locations, and transfer data between computers that are on different locations (possibly different locations in the same building, but the systems can also be

located in different buildings, cities, countries or on different continents). These facilities include computing power, information, programs, I/O-devices, etc. Long-haul computer networks and local area networks are designed to provide a large number of these facilities.

In this work we present a fundamental analysis of some important problems in distributed computing. In chapter 2 we consider the decentralized extrema-finding or election problems on rings of processors. In chapter 3 we consider a simple version of the load-distribution problem on rings of processors. In chapter 4 we give a class of controllers that avoid store-and-forward deadlock in packet switching networks. In chapters 5, 6 and 7 we make an extensive study of a notion of simulation of large processor networks on smaller networks.

This chapter is intended to provide the reader with some background in distributed computing (see also [vL83],[Ta81],[Co85]), and to give some definitions that will be needed in later chapters. After commenting on what we will understand by the terms "distributed computer system" and "distributed computing" in section 1.2., we discuss in section 1.3. three important types of distributed systems: long haul computer networks, local area networks and multi-processor architectures. Several control tasks or structures required in these networks will be analysed in later chapters of this work. In section 1.4. we mention some of the fundamental problems arising in distributed computing. In sections 1.5. to 1.7. we give a number of notions and notations used throughout this work.

1.2. What is distributed computing? There are no commonly accepted definitions of the terms "distributed computer system" and "distributed computing". (See e.g. [LL81].) A common feature of all distributed systems is that there are multiple processes, processors or computer systems, that communicate and cooperate with each other. Often a distinction is made between "parallel" and "distributed" computing. In a parallel computer system many identical processors will work on the same problem simultaneously, the processors work synchronously, and the communication delay time (of a message) is not large compared to the state transition time of a processor (the time needed for one internal computation step). The processors are connected in an "interconnection network". In a distributed computer system the processors (or computer systems) will work asynchronously, the processors may be of different types, and the communication delays may be large in comparison to the state transition time of a processor. The processors are connected in a "computer network".

We will mainly use the terms "distributed computer system" and "distributed computing" in a broad sense, including both parallel and distributed computing. Note that a distributed computer system may consist of several computer systems, and that some of them may be distributed systems themselves. This occurs, for instance, when a multi-processor computer is connected in a long haul computer network.

1.3. An overview of distributed computing. In this section we discuss three important types of distributed computer systems: long haul computer networks, local area computer networks and multi-processor machines. Several other types of distributed computing will not be discussed here, like data-flow computing (see e.g. [Bö84] or [Sh85]) and systolic computing (see e.g. [KvL83] or [MC80]). An appreciation of computer networks is required for the early chapters of this work. In later chapters we study aspects of multi-processor organizations and their interconnection networks.

1.3.1. Long haul computer networks. For various reasons it can be desirable to connect several computer systems situated at different geographic locations with each other. These systems then form a (long haul) computer network. In many cases the distances between the locations of the computer systems are large, as in e.g. world-wide networks.

With these networks, several new facilities are created:

(i) it is possible to send messages to other users at other computer systems via electronic mail.

(ii) it is possible to access, enter and modify data, stored at other computer systems.

(iii) it is possible to run (large) programs at another computer system. This is useful e.g. if the other system has (much) more computing power (e.g. it is a super computer), or to prevent problems with conversions of programs from one type of machine to another type.

There are more examples of the use of computer networks. Design and maintenance of these networks is often far from trivial. Functions, like the distribution of programs and data over the different machines, accounting, etc. must be provided by a distributed operating system. Either the distributed operating system can be placed on top of the -already existing- operating systems, running on the different machines in the network, or one can have a single network-wide distributed operating system (see e.g. [Mu85]). Many problems must be dealt with to guarantee a correct and efficient behavior of the network. Several of these problems will be mentioned in section 1.4. A good overview of existing network algorithms and architectures can be found in [Ta81]. (See also e.g. [Co85], [St85] or [Hl85].)

1.3.2. Local area networks. Another important class of distributed systems form the so-called "local area networks", (often abbreviated as: LAN's). The most significant difference between long haul networks and local area networks is the scale, which is much smaller for the latter. The smaller distances facilitate different technologies for faster information transfers between all sites involved. Usually many of the systems connected in a local area network are small work-stations, personal computers, etc, within a short range of distances. Typically all nodes of a local area network are located in the same building or in a few adjacent buildings, and the entire network is owned by one organization.

The advantages of a local area network over other (local) computer organizations, like a large mainframe with many terminals, are that users are given the flexibility and the ease of having their own personal computer, while also having access to facilities like the possibility to communicate with other users, shared databases, shared expensive I/O-devices among several users, etc.

One can also use a local area network to speed up algorithms by parallelization. By splitting an algorithm in smaller parts (possibly recursively) and letting different parts run on (possibly) different machines concurrently, one can speed up the time needed for carrying out the algorithm. Also for some problems one might want to design parallel algorithms specifically suited for this type of architecture. In this way one can use the otherwise unused capacity of the nodes of the local area network. This is useful because in many cases many of the nodes of a local area network are unused or under-utilized for a significant part of the time.

Often used current architectures for local area networks are the carrier sense multiple access network (e.g. ethernet) and the token ring network (see e.g. [Ta81]). Here we remark that a clear distinction must be made between the token ring network and the ring networks, discussed in chapter 2 and in chapters 5-7 of this work. The token ring network usually has at most one message transmitted at the time, and when a node receives a message, it can forward it simultaneously to the next node with a delay of only the time needed to send (or receive) one bit. In the ring networks many nodes can send a message simultaneously to their successor node on the ring, but a message can only be forwarded to the next node on the ring after it is received entirely by the current node. We also remark that some of the protocols, discussed in chapter 3 of this work can be implemented on a carrier sense multiple access network or a token ring network.

Many problems of distributed control, arising with long haul computer networks also arise with local area networks, often in a slightly different form and under different constraints. These problems, and other problems typical for local area networks have to be dealt with to ensure an efficient and correct behavior and a (near) optimal use of the local area networks. A partial list of these problems can be found in section 1.4. For a more detailed introduction in local area networks see e.g. [St84], [Ta81] or [Hl85].

1.3.3. Multi-processor networks. Conceptually it is only a small step from a local area network which allow fast communication between processes run on different machines in the network to machines with an architecture specifically designed for the parallelization of algorithms, the so-called "multi-processor" architectures. For many (large) problems, it may be very expensive or even impossible to build a single processor machine that solves these problems within reasonable time bounds. By using many processors (which may themselves be not very fast), and letting these processors share the work of the same problem, the problem may be solved much faster and/or cheaper.

A well-known classification of the multi-processor architectures is the classification in SIMD-machines and MIMD-machines, due to Flynn [F172]. Both types of machines consist of a large number of processors, that are able to communicate with each other in some way (e.g. by using an interconnection network or by means of shared memory). An SIMD-machine ("Single Instruction Multiple Data") is synchronized and in each timestep a subset of the processors carries out the same instruction (identified by the setting of a mask-bit), but may work on different data. An MIMD-machine ("Multiple Instruction Multiple Data") is not necessarily synchronized and each processor has it own sequence of instructions to carry it out. In general, (large) SIMD-architectures are easier to design and build, but, on the other hand, it is easier to implement parallel algorithms on an MIMD-machine.

There are various ways in which the communication between the processors can be provided. For instance it is possible to use a shared memory: processors are connected via a bus or a switching network with the same memory. A protocol is needed that resolves conflicts that arise when two or more processors try to "write" into the same memory location at the same moment, in an efficient way, or that avoids these conflicts.

Another important possibility is to connect the processors themselves in a network ("a multi-processor interconnection network"). The structure of such a network does not depend on geographic constraints (like in long haul networks), so it can and in general will be chosen such that the network is of a regular structure. In section 1.6. we will show some important network types. In section 1.4. we will also discuss some problems typical for multi-processor networks.

A good overview of different types of multi-processor architectures and algorithms running on these machines can be found in [vL85]. Also, many aspects of multi-processor machines are treated in [Pa84].

1.4. Problems in distributed computing. For a correct and efficient behavior and use of distributed systems, many problems must be dealt with. We will give a partial list of these problems, that only serves to give an idea of the type of problems that occur with distributed computing. With most problems we give one or a few references, that give further information on the problem and on known (partial) solutions to the problem. The list is not meant to be anywhere near complete. A much more extensive treatment of problems occurring with long haul computer networks and local area networks can be found in [Ta81]. We distinguish three classes of problems: problems dealing with "Cooperation", with "Communication" and with "Programming and Verification". In the corresponding sections we describe these notions more precisely. (The distinctions between the three classes is not very sharp.)

1.4.1. Problems dealing with "Cooperation". In most distributed systems, one (or perhaps more) process(es) can run on each processor, concurrently. The processes must cooperate to solve various tasks. We will mention some of the problems that (can) arise with this cooperation.

1. In some applications processes may "fork" off other processes, and start up other processes that are possibly located on other processors. Also, processes may be moved to other processors. As different processes may wish to communicate with each other, it must be administered what process is located at what processor. Also one may want the distribution of the processes over the processors to be "balanced", in the sense that each processor has about the same work-load. Another problem that arises in this area is that of detection that a computation has finished (the distributed termination detection problem, see [TvL86]).

2. Although a distributed system may be (and, in general, is) asynchronous, one still may want to have some notion of "the current time", for instance in order to be able to identify the state of the distributed system at some fixed moment. As clocks running at different processors may drift away from each other when they are not running at exactly the same speed, there must be some protocol to synchronize them or one must use some other mechanism to have the processors (virtually) synchronized. (See e.g. [La78] or [HSSD84].)

3. A notion of "the state of the system at some fixed moment in time" is useful, e.g. when a (part of) the system crashes. For a restart procedure one usually wants to restart from a consistent state of the system, corresponding to a state not long before the system crash, such that little work is lost. For such a restart procedure (and also in some other cases) it is often desirable to have one central processor (a "leader") that coordinates the restart procedure. In chapter 2 of this work we consider the problem of electing such a "leader" processor distributively in the case of a ring of processors.

4. For many problems in distributed computing one may want to have solutions that still behave correctly if one or a small number of processors and/or links go down. Also one may consider the case that a small number of processors work incorrectly or even behaves in an adverse way. The problem of reaching agreement in a network, where a small fraction of the processors may behave in an adverse manner is known as the Byzantine agreement problem. (See e.g. [LSP82] or [Re85].)

5. In general one wants to have a fair distribution of the workload over the processors, in terms of processor-use and/or memory-use. A special version of this load-distribution problem is addressed in chapter 3 of this work.

6. Processes may want to use the same resources at the same moment. Therefore one may want to have some process, called the "resource manager", that takes care of a fair and efficient allocation of resources to processes, and for instance avoids that processes must wait infinitely long for a requested resource. (See e.g. [RS82].)

1.4.2. Problems dealing with "Communication". In order to interact and cooperate with each other, processors must be able to communicate with each other. The various needs for communication give rise to a large number of problems. A few are mentioned below.

1. During the transmission of a sequence of bits over a transmission medium (e.g. a telephone cable or a satellite), some bits may be mutilated or disappear. This type of errors must be detected and eventually corrected. A protocol is needed to ensure correct transmissions over a single link. (See [Ta81], chap. 3.)

2. Also, large(r) sequences of bits may be mutilated or disappear, which means that a packet (of bits) must be sent again. In general, a message is split into smaller "packets", and these packets are transmitted individually. A protocol must ensure that each of the packets is received correctly (by the receiver), and that the packets are placed in the correct order. (See [SvL85] or [Ta81], chap. 4.)

3. In computer networks it is often the case that a message must be sent from some processor A (the "source" of the message) to some processor B (the "destination" of the message). In that case the message (or its corresponding packets) must follow some path through the network. This can be done either by setting up a virtual channel between A and B for the duration of the entire communication ("circuit switching"), which means that A and B can communicate over the virtual channel as if they have a direct link, or by sending each packet individually over the network ("packet switching" or "datagram service"). For a packet switching network the routes that the packets must take can be determined in different ways. The main distinction that can be made is between fixed routing strategies (which yield a fixed path between each pair of processors (A,B)) and variable routing strategies (in which packets can take different paths from the same source to the same destination, for instance to avoid areas of high message-traffic density). (See e.g. [Ta81], par. 5.2.)

4. A problem arising in packet switching networks is the problem of store-and-forward deadlock. Each processor has a limited number of buffers available to store packets that are moved from their source node

to their destination node. If some of the nodes in the network have no free buffers available and each of the packets stored at these nodes must be moved to another node in the set, then none of the packets can ever move. In chapter 4 we will discuss this type of deadlock and ways to avoid it.

5. If some shared communication medium is used, like a satellite, a radio channel or a bus, then collisions can occur (two or more processes try to send a message on the same moment), and protocols are needed that deal with collisions in an efficient way, or avoid them. (See e.g. [Ta81], chap. 6, or [St85], chap. 10.)

6. In multi-processor networks, the interconnection pattern of the processors has a strong influence on which algorithms can be implemented efficiently on the network. Therefore the choice of the (type of) interconnection pattern is a very important issue in the design of a multiprocessor network. (See e.g. [Wi81] or [BH83].)

1.4.3. Problems dealing with "Programming and Verification". An essential problem for distributed computing is how to design algorithms for it, and how to verify that a distributed algorithm indeed does what it is supposed to do. The corresponding problems for sequential algorithms are nowadays reasonably managed and a large number of techniques for designing and verifying sequential algorithms and programs exist. Due to the fact that many parts of a distributed algorithm will work concurrently, interact with each other, and influence each other in a not always predictable way, distributed algorithms seem more difficult to understand than sequential algorithms. Consequently distributed algorithms are also more difficult to design and verify. We now discuss some problems in this area.

1. For many problems the question arises: how can we solve this problem efficiently on some specific SIMD- or MIMD-machine? Often for an efficient use of the multi-processor system, it is necessary that one knows and uses specific characteristics of the architecture, like the size and type of the interconnection network. A notion, which might be a useful tool in writing this type of distributed algorithms, called

"emulation", is studied extensively in chapters 5, 6 and 7 of this work.

2. Also, for given problems, the question arises: what type of multi-processor architecture is best for this specific problem? What speed-up can we expect when we try to solve the problem on a multi-processor machine? Much experimental and theoretical work has already been done on problems of this type. (See e.g. [vV85] or [Ul84].)

3. As we have discussed above, verification techniques are needed to ensure that distributed programs indeed do what they are expected to do. There has been much study on this subject and a number of different techniques exists for proving properties of distributed programs. (See e.g. [AFdR80], [GdR84] or [Kn81].) Also, it is often difficult to debug a distributed program, also because of the parallelism (one has to follow a number of components simultaneously), and the nondeterminism, intrinsic to (asynchronous) distributed systems.

4. Presently there is only a relatively small number of programming languages for distributed systems (CSP [Ho78], OCCAM [Oc84], etc.). However, the availability of enough distributed programming languages that are easy to understand and use is essential for a wide-spread use of distributed systems (like multi-processor systems). Therefore, there is still a need for distributed programming languages, that are easy to use, available on a large number of systems and give efficient (parallel) code.

1.5. Notions and notations from set and graph theory. In this section we give some definitions and notations from set and graph theory that are used throughout this work. First we give some (well-known) notations from set theory.

Notations. Let S, T be sets, $k \in N$.

(i) $|S|$ denotes the number of elements in S (usually this notation is used for finite sets).

(ii) The set of subsets of S is denoted by $P(S)=\{T|T \subseteq S\}$.

(iii) The set of subsets of S with exactly k elements is denoted by $P_k(S) = \{T|T \subseteq S \wedge |T|=k\}$.

(iv) The Cartesian product of S and T is the set $S \times T = \{(s,t) \mid s\in S, t\in T\}$.

We expect the reader to be familiar with most elementary definitions from graph theory (cf. Berge [Bg76], Harary [Ha69]). If not mentioned otherwise, a graph is considered to be free from self-loops (i.e. $(u,v) \in E \Rightarrow u\neq v$), and parallel edges (i.e. $e_1 = (v,w) \in E \wedge e_2 = (v,w) \in E \Rightarrow e_1=e_2$). Unless stated otherwise we assume a graph to be undirected.

Definition. The path (or path-graph) with n nodes is the graph $P_n=(V^P_n,E^P_n)$ with $V^P_n = \{0,1,\ldots,n-1\}$ and $E^P_n = \{(i,j) \mid i,j\in V^P_n, |i-j|=1\}$. The directed path-graph with n nodes is the graph $\overrightarrow{P}^P_n = (V^P_n, \overrightarrow{E}^P_n)$ with $\overrightarrow{E}^P_n=\{(i,j) \mid i,j\in V^P_n, i\leq n-2, j=i+1\}$.

Definition. The complete graph with n nodes is the graph $K_n = (\{0,1,2,\ldots,n-1\}, \{(i,j) \mid i,j\in\{0,1,2,\ldots,n-1\}, i\neq j\})$.

Definition. The complement of a graph $G=(V,E)$ is the graph $G^C=(V,E^C)$ with $E^C = \{(v,w) \mid v,w\in V \wedge v \neq w \wedge (v,w)\notin E\}$.

Definition. For directed graphs $G=(V,E)$ let G^R be the directed graph obtained from G by reversing the direction of the edges, i.e., $G^R=(V,E^R)$ with $E^R = \{(w,v) \mid (v,w)\in E\}$.

Definition. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be graphs.

(i) The Cartesian sum of G and H is the graph $G+H = (V_G\times V_H, E_{G+H})$ with $V_G\times V_H = \{(v,w) \mid v\in V_G \wedge w\in V_H\}$ (the Cartesian product of sets V_G, V_H), and $E_{G+H} = \{((v_1,w_1),(v_2,w_2)) \mid (v_1,w_1),(v_2,w_2)\in V_G\times V_H$ and $\left[(v_1=v_2 \wedge (w_1,w_2)\in E_H) \text{ or } ((v_1,v_2)\in E_G \wedge w_1=w_2)\right]\}$.

(ii) The Cartesian product of G and H is the graph $G\times H = (V_G\times V_H, E_{G\times H})$ with $E_{G\times H}=\{((v_1,w_1),(v_2,w_2)) \mid (v_1,w_1),(v_2,w_2)\in V_G\times V_H$ and $(v_1,v_2)\in E_G$ and $(w_1,w_2)\in E_H\}$.

(iii) The normal product of G and H is the graph $G\bullet H = (V_G\times V_H, E_{G\bullet H})$ with $E_{G \bullet H} = \{((v_1,w_1),(v_2,w_2)) \mid (v_1,w_1),(v_2,w_2) \in V_G\times V_H$ and $\left[(v_1=v_2 \wedge (w_1,w_2) \in E_H)\right.$ or $((v_1,v_2) \in E_G \wedge w_1=w_2)$ or

$(v_1,v_2)\in E_G \wedge (w_1,w_2)\in E_H)\big]\}$.

(iv) The composition of G and H is the graph $G[H] = (V_G\times V_H, E_{G[H]})$ with $E_{G[H]}=\{((v_1,w_1),(v_2,w_2)) \mid (v_1,w_1),(v_2,w_2)\in V_G\times V_H$ and $\big[(v_1,v_2)\in E_G$ or $(v_1=v_2 \wedge (w_1,w_2)\in E_H)\big]\}$.

Definitions (i),(ii) and (iii) were taken from Berge[Bg76]; definition (iv) was taken from Harary[Ha69]. (Note that the definition of [Ha69] of the product of two graphs, G×H, equals our definition of the Cartesian sum of two graphs, G+H.)

With $d_G(b,c)$ for $b,c \in V$ we denote the distance of b to c in the graph G=(V,E), i.e., the length of the shortest (directed) path from b to c. If G is clear from the context, we drop the subscript G.

<u>Definition</u>. Let G=(V,E) be an undirected graph and let $n=|V|$. A linear ordering of V is a bijection $f\colon V \rightarrow V^{P}_{n}$. f is said to have bandwidth K if $K= \max \{|f(u)-f(v)| \mid (u,v)\in E\} = \max \{d_{P_n}(f(u),f(v)) \mid (u,v)\in E\}$, i.e., the maximum distance in P_n between the image of adjacent nodes in G is K. The <u>bandwidth</u> of G is the minimum bandwidth over all linear orderings of V, i.e., Bandwidth(G) = min {bandwidth(f) | f is a linear ordering of V}.

For the notion of cyclic bandwidth, the nodes are mapped to a ring, instead of to a path. The definition of the ring network with n nodes R_n will be given in section 1.6.1. The notion of cyclic bandwidth was introduced in [LVW84]. The notion of directed cyclic bandwidth is new.

<u>Definition</u>. Let G=(V,E) be an undirected graph. A linear ordering f is said to have cyclic bandwidth K if $K= \max \{d_{R_n}(f(u),f(v)) \mid (u,v)\in E\}$. The <u>cyclic bandwidth</u> of G is the minimum cyclic bandwidth over all linear orderings of V, i.e., Cyclic Bandwidth(G)= min {Cyclic Bandwidth(f) | f is a linear ordering of V}.

For directed (cyclic) bandwidth, if $(u,v) \in E$ then we want to have a path of length at most K from f(u) to f(v) in $\vec{P}_n$ ($\vec{R}_n$).

Definition. Let G= (V,E) be a directed graph. A linear ordering f is said to have directed bandwidth K, if for all $(u,v) \in E$ $f(v) > f(u)$, and $K = \max \{d_{\vec{P}_n}(f(u),f(v)) \mid (u,v) \in E\}$. f is said to have directed cyclic bandwidth K if $K= \max \{d_{\vec{R}_n}(f(u),f(v)) \mid (u,v) \in E\}$. If there exists a linear ordering with directed bandwidth K and no linear ordering with a smaller directed bandwidth then we write Dir Bandwidth(G) = K. Similarly we write Dir Cyclic Bandwidth(G) = K if K is the least possible cyclic bandwidth for all linear orderings of V.

Note that there only exists a K with Dir Bandwidth(G) = K, iff G is cycle-free: suppose G contains a cycle with nodes $v_1, v_2, \ldots v_e$ and f is a linear ordering with directed bandwidth K for some arbitrary K, then $f(v_1) < f(v_2) < \ldots < f(v_e) < f(v_1)$, contradiction. If G is cycle-free, then there exists a linear ordering f of V, such that $(u,v) \in E \Rightarrow f(u) < f(v)$. Now Dir Bandwidth(f) $\leq |V| - 1$.

We now introduce some notations that are needed to denote strings and sets of strings, etc. First we introduce some notations that will be used when dealing with bitstrings:

$\frac{o}{1}$: a bit that can be o or 1

$\bar{\alpha}$: the complement of bit α ($\bar{o} = 1$, $\bar{1} = o$)

$\bar{b}$: the address one obtains by complementing every bit of b ($\overline{b_1 \ldots b_n} = \bar{b}_1 \ldots \bar{b}_n$)

$\alpha \equiv \beta$: the 'equivalence' test on bits ($O{\equiv}O = 1$; $O{\equiv}1 = O$; $1{\equiv}O = O$; $1{\equiv}1 = 1$)

b : the n-bit address $b_1 \ldots b_n$

$b|_i$: $b_1 \ldots b_i$ (truncation after the i^{th} bit)

${}_i|b$: $b_i \ldots b_n$ (truncation "before" the i^{th} bit)

$(b)_i$: b_i (the i^{th} bit).

For functions f defined on n-bit numbers b we use :

$f_i(b)$: $(f(b))_i$ (projection on the i^{th} bit)

We use b, c, ... to denote full addresses and x, y, ... to denote segments of bits. Individual bits are denoted α, β, ... We also use the following notations:

[o] : zero or one occurrence of bit o (i.e., "empty" or "o")

[1] : zero or one occurrence of bit 1 (i.e., "empty" or "1")

(o1)* : zero or more repetitions of the string o1 (as required)

(1o)* : zero or more repetitions of the string 1o (as required)

The length (n) of a bitstring will always be clear from the context, and is usually not given by separate indices. For example, the notation (o1)*[o] for n odd will denote the string $(o1)^{\lfloor n/2 \rfloor}o$. For n even it will denote the string $(o1)^{n/2}$.

1.6. <u>Important interconnection networks</u>. In this section we introduce some important networks (graphs), commonly used as (proposals for) interconnection patterns of processors in processor networks.

1.6.1. <u>The ring network</u>. One of the most elementary interconnection patterns is the ring. The ring hardly occurs as an interconnection pattern for multi-processor networks, but often is used in local area networks (cf. Tanenbaum [Ta81]). The nodes in a ring R_n are named 0,1,...,n-1 in consecutive order (cf fig 1.6.1.1).

<u>Definition</u>. The ring network (or n-ring) is the graph $R_n = (V_n^R, E_n^R)$ with $V_n^R = \{0,1,\ldots,n-1\}$ and $E_n^R = \{(i,j) \mid i,j \in V_n^R, |i-j|=1 \text{ or } |i-j|=n-1\}$. The unidirectional ring (network) with n nodes is the directed graph

$\vec{R}_n = (\vec{V}_n^R, \vec{E}_n^R)$ with $\vec{V}_n^R = V_n^R$ and $\vec{E}_n^R = \{(i,i+1) | i \in \vec{V}_n^R\}$, where + is the addition modulo n.

1.6.2. The two-dimensional grid network. The two-dimensional grid (or mesh) has been used as a processor interconnection network, and extensive studies have been made of algorithms to be executed on a grid (e.g. Nassimi & Sahni [NS80]). We use a version of the grid with "wrap-around" connections along the boundaries and a version without wrap-around connections. Let GR_n be the nxn grid network (with n^2 nodes), with wrap-around connections, and let GR'_n be the nxn grid network without wrap-around connections. The nodes of GR_n and GR'_n are named by their plane coordinates (i,j) with $0 \leq i,j \leq n-1$ in the usual representation of the nxn grid. (Cf. fig. 1.6.2.1.)

Definition. The two-dimensional grid network with wrap-around connections is the graph $GR_n = (V_n, E_n)$ with $V_n = \{(i,j) | i,j \in N$ and $0 \leq i,j \leq n-1\}$ and $E_n = \{((i,j), (i',j')) | (i,j), (i',j') \in V_n$ and $(i=i' \wedge j=(j'\pm 1) \bmod n)$ or $(i=(i'\pm 1) \bmod n \wedge j=j')\}$. The two-dimensional grid without wrap-around connections is the graph $GR'_n = (V_n, E'_n)$, with $E'_n = \{((i,j), (i',j')) | (i,j), (i',j') \in V_n$ and $(i=i' \wedge j=j'\pm 1)$ or $(i=i'\pm 1 \wedge j=j')\}$.

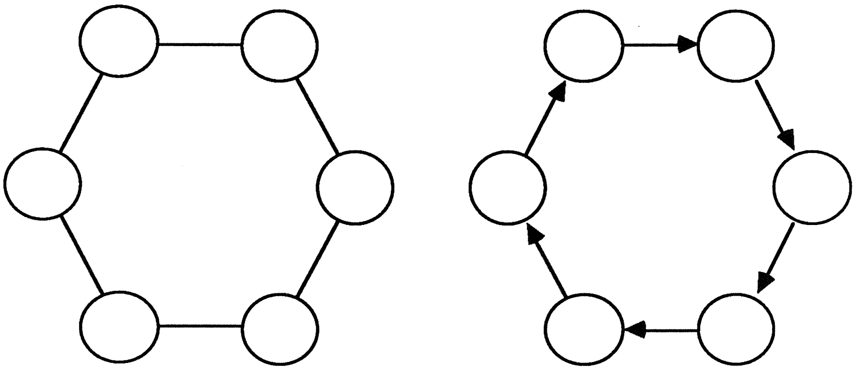


fig. 1.6.1.1. a) R_6, b) $\vec{R}_6$.

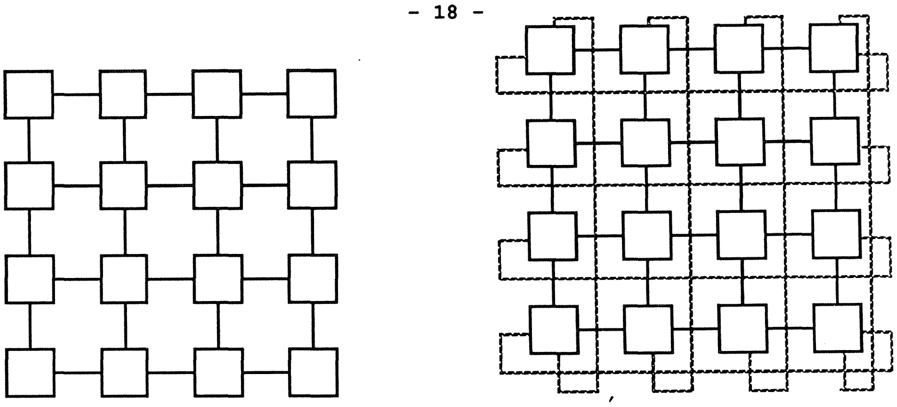


fig. 1.6.2.1. a) GR_4. b) GR'_n.

1.6.3. The shuffle-exchange network and the 4-pin shuffle. Stone [St71] proposed a network, called the shuffle exchange network, which has been successfully used as the interconnection network underlying a variety of parallel processing algorithms. However, there are two slightly different types of graphs, both realizing Stone's concept of a shuffle exchange network. We will use the terminology of [FF82] and call these graphs the shuffle-exchange graph and the 4-pin shuffle, respectively. The nodes of the shuffle-exchange graph and the 4-pin shuffle are given n-bit addresses in the range o...2^n-1. In the shuffle-exchange graph there is an edge from node b to node c if and only if b can be "shuffled" (move the leading bit to tail position) or "exchanged" (flip the tail bit) into c. In the 4-pin shuffle there is an edge from node b to node c if and only if c can be reached from b by a shuffle or by a shuffle followed by an exchange. Computations proceed by iterating the networks some n or more times in a synchronized manner. We use the notation SE_n and S_n to denote the shuffle-exchange graph and the 4-pin shuffle, respectively, with 2^n nodes. The inverse shuffle exchange graph ISE_n and the inverse 4-pin shuffle IS_n are obtained by reversing the direction of the edges in the shuffle exchange graph and the 4-pin shuffle, respectively. We use the notations from section 1.5.

Definition. The shuffle-exchange network is the directed graph with self-loops $SE_n = (V_n^{SE}, E_n^{SE})$ with $V_n^{SE} = \{b_1 \ldots b_n \mid \forall 1 \leq i \leq n \; b_i = \frac{o}{1}\}$ and $E_n^{SE} = \{(b,c) \mid b, c \in V_n$ and $((\forall 2 \leq i \leq n \; b_i = c_{i-1} \wedge b_1 = c_n)$ or $(\forall 1 \leq i \leq n-1 \; b_i =$

$c_i \wedge b_n = \overline{c}_n)\}$. The 4-pin shuffle network is the directed graph with self-loops $S_n = (V_n^S, E_n^S)$ with $V_n^S = V_n^{SE}$ and $E_n^S = \{ (b,c) \mid b,c \in V_n$ and $\forall 2 \le i \le n\ b_i = c_{i-1} \}$. The inverse shuffle-exchange network is the directed graph with self-loops $ISE_n = SE_n^R$. The inverse 4-pin shuffle is the directed graph with self-loops $IS_n = S_n^R$.

It follows that in SE_n a node $b_1 \ldots b_n$ is connected to $b_2 \ldots b_n b_1$ and $b_1 \ldots b_{n-1} \overline{b}_n$, and in S_n it is connected to $b_2 \ldots b_n o$ and $b_2 \ldots b_n 1$.

1.6.4. The cube network. The cube network with 2^n nodes (also called an n-cube) has perhaps been the first proposal ever for processor interconnection. The nodes in the network again are given n-bit addresses in the range $o \ldots 2^n-1$, and there is an edge from node b to node c if and only if c is obtained by flipping precisely one bit in b. Information can be routed from a source b to a destination c in at most n steps, by flipping the bits b_i to the corresponding bits c_i in some order. Since nodes thus have degree n, the cube network is considered practical only for small values of n. The i^{th} bit of an address b is denoted by b_i $(1 \le i \le n)$. For bitstrings x,y with $|x| = |y|$, let d(x,y) be the Hamming distance between x and y, i.e., the number of bit-positions in which x and y differ. (See, for example, Deo [De74] sect. 12-5.)

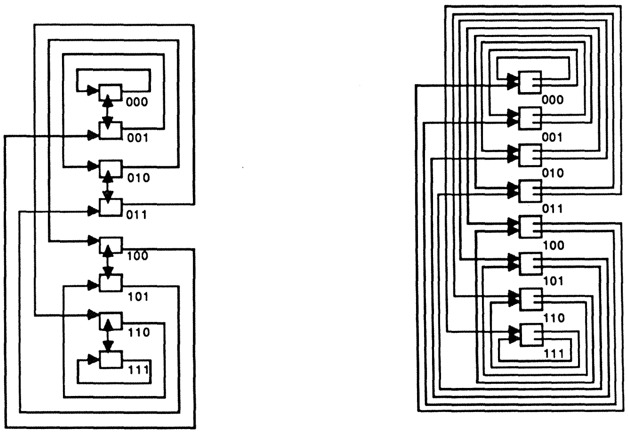


fig. 1.6.3.1. a) SE_8. b) S_8.

Definition. The cube network (or n-cube) is the graph $C_n = (V_n^C, E_n^C)$ with $V_n^C = \{ (b_1 \ldots b_n) \mid \forall\ 1 \le i \le n{:}\ b_i = \frac{o}{1} \}$ and $E_n^C = \{ (b,c) \mid b,c \in V_n$ and $d(b,c) = 1 \}$.

1.6.5. The cube-connected cycles. Another important network, proposed for processor interconnection by Preparata and Vuillemin [PV81] is the cube-connected cycles network. For denoting the 'cube-connected cycles' graph we use the notation introduced in [FF82]. Processors in the cube connected cycles with $r.2^r$ nodes are addressed by r-bit strings with a "divide" in any one position between bits. The position to the left of the first bit is identified with the position to the right of the last bit, so there are r positions for the divide. A node $p_1 \ldots p_{i-1} |\ p_i \ldots p_r$ is connected to the following 3 nodes:

$p_1 \ldots \overline{p_{i-1}} |\ p_i \ldots p_r$

$p_1 \ldots p_{i-2} |\ p_{i-1}\ p_i \ldots p_r$

and $p_1 \ldots p_{i-1}\ p_i |\ p_{i+1} \ldots p_r$.

Definition. The cube-connected cycles graph with 2^r nodes is the (undirected) graph $CCC_r = (V_r^C, E_r^C)$, with $V_r^C = \{p_1 \ldots p_{i-1} | p_i \ldots p_r \mid p_1 \ldots p_r \in (\frac{0}{1})^r,\ 1 \le i \le r\}$ and $E_r^C = \{ (p_1 \ldots p_{i-1} | p_i \ldots p_r,\ p_1 \ldots \overline{p_{i-1}} | p_i \ldots p_r) \mid (p_1 \ldots p_{i-1} | p_i \ldots p_r) \in V_r\} \cup \{ (p_{1 \ldots} p_{i-1} | p_i \ldots p_r,\ p_1 \ldots p_i | p_{i+1} \ldots p_r) \mid p_1 \ldots p_{i-1} | p_i \ldots p_r,\ p_1 \ldots p_i | p_{i+1} \ldots p_r \in V_r\}$. (Let - (+) be the subtraction (addition) modulo r.)

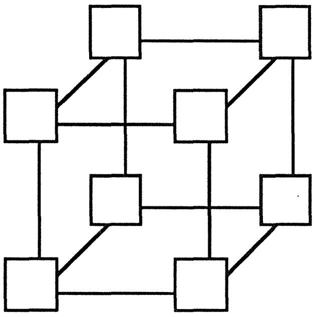


fig. 1.6.4.1. C_3.

1.6.6. We will drop the superscripts R, C, CCC, etc. from V_n^R, E_n^R,etc., when it is clear from the context what type of network is used.

1.7. Notions and notations from complexity theory. In this section we introduce some (well-known) notations for estimating bounds on (integer) functions and give a very brief introduction to the theory of NP-completeness.

Definition. Let f, g be functions $N \rightarrow N$.
(i) g(n) is said to be O(f(n)), if and only if there exists a constant $c \in R^+$, such that $g(n) \leq c \cdot f(n)$ for all but finitely many values of n.
(ii) g(n) is said to be $\Omega(f(n))$, if and only if there exists a constant $c \in R^+$, such that $g(n) \geq c \cdot f(n)$ for all but finitely many values of n.
(iii) g(n) is said to be $\Theta(f(n))$, if and only if $g(n)=O(f(n))$ and $g(n)=\Omega(f(n))$.

We now give a very brief introduction to the theory of NP-completeness. An extensive treatment of the subject can be found in [GJ79].

We call a problem a decision problem, if with an input of a certain type (for instance a graph, or a set of integers), called the "instance" of the problem, it must output a boolean which denotes whether a certain property of the input holds or not. (An example would be: is the input-graph connected, or: does the inputgraph contain a Hamiltonian path.) The set of decision problems that are solvable on some standard deterministic model of computation, like a deterministic Random Access Machine or a Turing Machine, in time polynomial in the size of the input is denotes as P.

Next consider non-deterministic computations. In a non-deterministic computation, non-deterministic choices from finitely many alternatives can be made a number of times. In a non-deterministic step the computation can proceed in different ways, depending on the choice that is made. We say that the entire computation yields the output true, if and only if the non-deterministic choices can be made in such a way that the output true is yielded by a computation (i.e., there is at least one possible computation yielding true). The time needed for the entire non-deterministic computation is the maximum time needed over all possible computations. The class of decision problems solvable on a (standard) Random Access Machine or Turing machine in time polynomial in the size of the input with a non-deterministic computation, is denoted as NP.

It is a long standing open problem to determine whether P=NP or P≠NP, i.e., whether all problems in NP can be solved with a deterministic algorithm in polynomial time or not. In general it is conjectured that P≠NP. (It is obvious that $P \subseteq NP$ holds.)

Definition. A decision problem A is called polynomial time reducible to a decision problem B, if there exists a function f, that maps instances of problem A to instances of problem B, such that

(i) f can be computed in polynomial time

(ii) for all instances X of problem A, X yields the answer yes with regard to problem A, if and only if f(X) yields the answer yes with regard to problem B.

(Compare with [GJ79, p.111].) A problem is called NP-hard, if every problem in NP is polynomial time reducible to it. It is called NP-complete, if it is NP-hard and a member of NP. It follows that if an NP-hard or NP-complete problem A is solvable in polynomial time, then this would mean that P=NP, and hence every problem in NP were solvable in polynomial time.

In 1971 Cook [Co71] showed that there indeed exist "hardest problems in NP", by proving the SATISFIABILITY problem NP-complete. SATISFIABILITY is the following problem: given a boolean expression in conjunctive normal form over boolean variables $b_1, \ldots b_n$, decide whether one can assign to each variable b_i a value from {true,false}, such that the value of the expression is true. After this result was obtained, a large number of other problems in computing were proven to be NP-complete. For our purposes, important examples are HAMILTONIAN PATH (given a graph G, does G contain a Hamiltonian path), HAMILTONIAN CIRCUIT, and the directed variants of these. Other problems that are known to be NP-complete will be introduced where necessary.

In general, one proves the NP-completeness of a problem by showing that the problem is a member of NP and showing that another NP-complete problem is polynomial time reducible to it.

CHAPTER TWO

DECENTRALIZED EXTREMA-FINDING
IN A RING OF PROCESSORS

2.1. Introduction. Consider a ring of n processors, distinguished by unique identification numbers. In general it is assumed that the size n of the ring is not known to the processors. There is no central controller. The problem is to design a distributed algorithm for finding the processor with the highest number, using a minimum number of messages. The elected processor can act as a 'leader' (central controller). Every processor (possibly several or all processors simultaneously) can start the "election", and every processor has to use the same algorithm. We further assume that the processors work fully asynchronously and cannot use clocks and/or timeouts. (Frederickson and Lynch [FL84], and Vitanyi [Vi84] analyzed the case where this strict assumption of asynchronicity does not hold, and show that in that case a significantly smaller number of messages is needed, if one is willing to spend considerably more time.) This latter assumption makes that we can assume that the algorithm is message-driven : except for the initialization of an election a processor can only perform actions upon receipt of a message. We also assume that there are no faulty processors and no faults in the communication subsystem.

The leaderfinding or "election" problem has received considerable attention, after it was proposed by Le Lann [LL77] in 1977. It has been studied for unidirectional rings as well as for general, bidirectional rings. Figures 2.1.1. and 2.1.2. summarize the solutions presently known for both cases, together with the worst-case or average number of messages required for each algorithm. (All logarithms are taken to the base 2, unless stated otherwise.)

For most of the bidirectional algorithms of figure 2.1.2. and also for the algorithms given in section 2.2. the assumption of a global sense of orientation (i.e., each processor knows the left and right

Algorithm	Average	Worst-Case
Le Lann (1977)	n^2	n^2
Chang & Roberts (1979)	$n H_n$ *	$0.5 n^2$
Peterson (1982)	0.943 n log n *	1.44.. n log n
Dolev, Klawe & Rodeh (1982)	0.967 n log n *	1.356 n log n

Fig. 2.1.1. Election Algorithms for Undirectional Rings
*: Result experimentally obtained by Everhardt [Ev84].

direction on the ring) is unnecessary. In our lowerbound analysis we do assume a global sense of orientation for bidirectional rings. This only strengthens our lowerbound results.

In this chapter we will derive some upperbound results, as well as some lowerbound results on the decentralized extrema finding problem. In section 2.2. we improve on the known upperbound for the average number of messages that is needed for decentralized extrema finding in a bidirectional ring. In section 2.3. we give various lowerbound results on the decentralized extrema finding problem in rings of processors.

* H_n is the n'th harmonic number, i.e., $H_n = \sum_{i=1}^{n} \frac{1}{n} \approx 0,69 \log n$.

Algorithm	Average	Worst-Case
Gallager et al. (1979)		5 n log n
Hirschberg & Sinclair (1980)		8 n log n
Burns (1980)		3 n log n
Franklin (1982)		2 n log n
Korach, Rotem, & Santoro (1981)	(prob.) 0.70..n H_n	(prob.) 0.5 n^2
Santoro, Korach & Rotem (1982)		1.89 n log n
Bodlaender & van Leeuwen (1985)	(det.) 0.70..n H_n	(det.) 0.25 n^2
Van Leeuwen & Tan (1985)		1.44.. n log n
Moran, Shalom & Zaks (1985)		1.44.. n log n

Fig. 2.1.2. Election Algorithms for Bidirectional Rings

2.2. New upperbounds for decentralized extrema-finding in a ring of processors.

2.2.1. Introduction. In 1981 Korach, Rotem, and Santoro [KRS81] gave a probabilistic algorithm for decentralized extrema-finding in bidirectional rings that uses a smaller (expected) average number of messages than any deterministic algorithm for the problem in unidirectional rings requires. In this section we consider the key question of whether decentralized extrema-finding can be solved more efficiently in bidirectional rings than in unidirectional rings by a deterministic algorithm. (The question was first posed by Pachl, Korach, and Rotem [PKR82], who proved a lowerbound of nH_n on the average number of messages required by any reasonable algorithm for leader-finding in unidirectional rings.)

Consider a ring of n processors with identification numbers X_1 through X_n. Without loss of generality we may assume each X_i to be an integer between 1 and n, hence $X \equiv X_1 X_2 \ldots X_n$ is a permutation. We also assume that f is "random", i.e., we assume that every permutation can occur with an equal probability of $\frac{1}{n!}$. One technique of decentralized extrema-finding in bidirectional rings makes use of the "peaks" in a circular permutation. Assume that initially all processors are active.

Definition. A peak in a ring of active and non-active processors is an active processor X_i that is larger that the active processors immediately to the left and to the right of X_i, assuming a fixed clock-wise orientation of the ring.

A typical algorithm due to Franklin [Fr82] operates in the following way. During one stage of the algorithm all active processors X_i send their identification number to the nearest active processors to the left and to the right. (Intermediate, inactive processors simply relay messages onwards.) When an active processor finds out that it has a larger active "neighbor" to the left or to the right, it becomes non-active. It is clear that in one stage only 2n messages need to be exchanged, and that precisely the peaks of the current permutation pass on to the next stage. As the number of peaks is not larger than half the number of currently active processors, Franklin's algorithm requires at most log n stages and (hence) 2n log n messages. The experimentally observed, smaller number of messages on the average in Franklin's algorithm might be explained as follows.

Theorem 2.2.1.1. (Bienaymé [Bi1874], 1874). The average number of peaks and troughs in a permutation of n elements is $\frac{1}{3}(2n-1)$.

It follows that one stage of Franklin's algorithm will leave about $\frac{1}{3}n$ processors ("peaks") active on the average. Assuming that the order type of the resulting configuration is again random, repetition shows that Franklin's algorithm requires only $\log_3 n$ stages and hence $2n \log_3 n \approx 1.26\, n \log n$ messages on the average.

In another technique of decentralized extrema-finding, identification numbers are send on the ring in some direction and travel until a processor is encountered that "knows" that the passing identification number cannot be the largest. In a typical algorithm due to Chang and Roberts [CR79] identification numbers are all send in the same direction and are annihilated by the first larger processor that is encountered. Thus all identification numbers except the largest are annihilated on their way around the ring, and the "leader" is identified as the only processor that eventually receives its own identification number as a message again. Knowing it is elected, the leader will send its identification number around the ring in another n messages to inform the processors of the result.

Given a random sequence (e.g. a time-series), an "upper record" is any element that is larger that all the preceding ones. The study of records was pioneered by Chandler [Ch52] in 1952, as part of the general theory of "order statistics" (see e.g. Galambos [Ga78], Sect. 6.3). Let X be a random sequence. Let $v_o=1$, and let v_i be the index of the first upper record with index larger than v_{i-1} $(i\geq 1)$. Thus v_i is a random variable for the position of the i^{th} upper record in the sequence. It is well known that the distribution of each v_i does not depend on the distribution function of the elements of the random sequence (cf. Galambos [Ga78], lemma 6.3.1) and that we may assume in fact that the elements are uniformly distributed. Observe that v_1 is the distance to the 'first' upper record of the sequence. The following result repeatedly occurs in the theory (see e.g. [Ch52], [FS54], [HW73]).

Theorem 2.2.1.2. The average distance to the first upper record in a random sequence of length n is $H_n-1 \approx 0.69 \log n$.

Proof. (sketch).

One can show that $P(v_1=j) = \frac{1}{j(j-1)}$ $(j\geq 2)$. Thus the average distance to v_1 is equal to

$$\sum_{j=2}^{n} \frac{j-1}{j(j-1)} = \sum_{j=2}^{n} \frac{1}{j} = H_n-1 = 0.69 \log n. \quad \square$$

The theory of record distributions in random sequences was considerably advanced by Renyi [Rn62] in 1962. He also derived the following useful fact, proved later by David and Barton [DB62] (p.181) by a combinatorial argument.

Theorem 2.2.1.3. For every $k \geq 1$ and $1 < j_1 < \ldots < j_k$ one has that

$$P(v_1 = j_1; \ldots; v_k = j_k) = \frac{1}{j_k \cdot \prod_{i=1}^{k} (j_i - 1)}.$$

Renyi [Rn62] proved that the number of upper records in a random permutation of n elements has the same distribution as the number of cycles in a random permutation. It follows from results of Feller [Fe45] from 1945 that this number is normally distributed, with expected value $H_n \approx 0.69 \log n$ (see also [Rn62]).

The results from the theory of order statistics apply to decentralized extrema-finding by observing that e.g. in the algorithm of Chang and Roberts the message generated by an X_i is propagated to the first upper record in the random sequence $X_i X_{i+1} \ldots$. (Because the message can travel all the way around the ring, the sequence is considered to have length n.) By theorem 2.2.1.2. a message will travel over H_n links "on the average", before it is annihilated. It follows that the algorithm of Chang and Roberts uses $nH_n \approx 0.69\, n \log n$ messages on the average. (This fact was proved by Chang and Roberts [CR79] without reference to the theory of order statistics.) By a result of Pachl, Korach, and Rotem [PKR82] the algorithm is optimal for unidirectional rings. In this section we will show that the algorithm is not optimal for bidirectional rings, i.e., bidirectional rings are "faster".

The section is organized as follows. In section 2.2.2. we review a probabilistic algorithm for decentralized extrema-finding due to Korach, Rotem, and Santoro [KRS81] and derive a deterministic algorithm for the problem that uses only $\frac{3}{4} nH_n \approx 0.52\, n \log n$ messages on the average. In section 2.2.3. we improve the analyses to obtain a bound about $0.7nH_n \approx 0.48\, n \log n$ messages for both algorithms.

2.2.2. Decentralized extrema-finding in a bidirectional ring using a small number of messages on the average. We begin by describing a probabilistic algorithm for extrema-finding in a bidirectional ring due to Korach, Rotem, and Santoro [KRS81] that uses an "expected" number of $\frac{3}{4}nH_n$ messages. We subsequently derive a deterministic algorithm for the problem that uses the same number of messages on the average (over all rings of n processors).

The probabilistic algorithm employs the second technique described in section 2.2.1 but, instead of all X_i sending their identification number in the same direction on the ring like in Chang and Roberts' method, the processors randomly decide to send their identification number to the left or to the right. With messages going clockwise and counterclockwise on the ring, it is expected that many messages run into "larger" messages and (hence) are annihilated sooner, thus resulting in the smaller message complexity of the algorithm. The algorithm in every processor consists of three successive stages, as described below.

```
Algorithm-P
Each processor $X_i$ keeps the largest identification number it has
seen in a local variable $MAX_i$ ($1 \le i \le n$). Each processor $X_i$ goes
through the following stages.

Stage 1 (initialization)
$MAX_i := X_i$;
choose a direction $d \in$ {left, right} with probability $\frac{1}{2}$;
send message $<X_i>$ in direction d on the ring;

Stage 2 (election)
repeat the following steps, until the end of the election is sig-
naled by receipt of a <!> message:
if two messages are received from the left and the right simultane-
ously, then ignore the smaller message and proceed as if only the
larger message is received;
if message $<X_j>$ is received from a neighbor, then
if $X_j$>$MAX_i$ then $MAX_i$ := $X_j$;
                pass messages <$X_j$> on
elif $X_j$ = $MAX_i$ then {$X_i$ has won the election}
                send message <!> on the ring
fi;
```

Stage 3 (inauguration)
if a message <!> is received, the election is over and MAX_i holds the identification number of the leader;
if this processor was elected in stage 2 then the inauguration is over, otherwise pass message <!> on and stop.

One easily verifies that a processor X_i wins the election if and only if its identification number succeeds in making a full round along the ring in a direction chosen in stage 1. Thus, at the moment that a unique processor X_i finds out that it is the leader, all processors must have set their local MAX-variable to X_i. It follows that it is sufficient to send a simple <!> message around the ring for an inauguration and as a signal that the election is over and that the algorithm is correct. We assume that all processors start the election simultaneously, otherwise the first message a processor receives serves to wake it up and trigger its stage 1, before it actually processes the message. For the analyses we will assume that the processors work synchronously.

Theorem 2.2.2.1. (Korach, Rotem, and Santoro [KRS81]).

(i) Algorithm-P uses $\approx \frac{1}{2}n^2$ messages in the worst case,

(ii) Algorithm-P uses (at most) $\approx \frac{3}{4}nH_n \approx$ o.52 n log n messages in the expected case.

Proof.

(i) The worst case occurs in a ring X≡n n−1...2 1, when all processors decide to send their identification numbers to the right (as in the algorithm of Chang and Roberts [CR79]). The number of messages adds up to $\frac{1}{2}n(n-1)+n \approx \frac{1}{2}n^2$.

(ii) Observe that the message generated by X_i (in stage 1) will be annihilated by the first upper record in the chosen direction on the ring. If the first upper record had decided to send its identification number in the opposite direction, i.e., towards X_i, then the messages meet "half way" and the $<X_i>$-message is killed right there. There is probability $\frac{1}{2}$ that the $<X_i>$-message needs to travel only half the distance to the first upper record in either direction on the ring. Using theorem 2.2.1.2., the expected number of $<X_i>$-messages will be $\frac{1}{2}H_n+\frac{1}{2}\cdot\frac{1}{2}H_n = \frac{3}{4}H_n$. (In case the first upper record decides to send its identification number away from X_i, it is possible that the second upper record decides to send its identification number towards X_i. If this happens it will kill the message of the first upper record, and it can conceivably stop the $<X_i>$-message even before reaching the position of the first upper record. Thus the expected number of messages will be slightly less than $\frac{3}{4}H_n$ per processor, cf. section 2.2.3.) It follows that the total number of messages exchanged is less than $\frac{3}{4}nH_n+n \approx \frac{3}{4}nH_n \approx 0.52$ n log n in the expected case. □

Observe that Algorithm-P is probabilistic and, hence, no proof in itself that decentralized extrema-finding is more efficient for bidirectional rings than for unidirectional rings. To resolve the problem we devise a version of Algorithm-P in which stage 1 is replaced by a purely deterministic step. The idea is to let a processor X_i send its $<X_i>$-message in the direction of its smallest neighbor, instead of letting it decide the initial direction by random choice. If X_i is beaten by a neighbor right away in the first exchange, it is made "inactive" for the remainder of the election.

Algorithm-D

Similar to Algorithm-P, except that for each processor X_i stages 1 and 2 are replaced as follows:

Stage 1*

send message $<*X_i>$ to both neighbors on the ring;

wait for the messages $<*X_{i-1}>$ and $<*X_{i+1}>$ of both neighbors (with the indices "i-1" and "i+1" interpreted in the usual circular sense as indices of the left and right neighbor, resp.);

```
MAX$_i$ := max($X_{i-1}$, $X_i$, $X_{i+1}$);
if MAX$_i$ = $X_i$ then
      if $X_{i-1} < X_{i+1}$ then send messages <$X_i$> to the left
                         else send message <$X_i$> to the right
      fi
fi;
```

Stage 2* (election)

repeat the following steps, until the end of the election is signaled by receipt of a <!> message:

if two messages are received from the left and the right simultaneously, then ignore the smaller message and proceed as if only the larger message is received;

if message <X_j> is received from a neighbor, then

```
if $X_j$>MAX$_i$ then MAX$_i$ := $X_j$;
                  pass message <$X_j$> on
elif $X_j$ = MAX$_i$ and $X_i$ = $X_j$
                  then {$X_i$ has won the election}
                  send message <!> on the ring
elif $X_j$ = MAX$_i$ and $X_i \neq X_j$
                  then {the neighbor of $X_i$ will win the election}
                  pass message <$X_j$> on
fi;
```

(Stage 3 is unchanged.)

Algorithm-D is correct by the same argument as used for Algorithm-P. Note that in Algorithm-D, stage 1* uses only 2n messages and eliminates at least $\frac{1}{2}$n processors from active participation in the election. The active processors that remain and send an <X_i>-message on the ring, will always have an inactive neighbor to the left and to the right.

Theorem 2.2.2.2

(i) Algorithm-D uses $\approx \frac{1}{4}n^2$ messages in the worst case,

(ii) Algorithm-D uses (at most) $\approx \frac{3}{4}nH_n \approx 0.52\ n \log n$ messages in the average case.

Proof.

(i) At most $\frac{1}{2}n$ processors are still active after stage 1*, and the active processor are separated by at least one inactive processor. Suppose the largest processor sends its identification number to the right. The worst case occurs if every second processor sends its identification number in the same direction and is not annihilated before it reaches the largest. This generates at most $n + \sum_{i=1}^{n/2} (n-2i) \approx \frac{1}{4}n^2$ messages. The worst case occurs in a ring of the form $X \equiv n\ 1\ n-1\ \lceil \frac{1}{2}n \rceil\ n-2\ \lceil \frac{1}{2}n \rceil - 1 \ldots$ (the shuffle of $n\ n-1\ \ldots\ \lceil \frac{1}{2}n \rceil + 1$ and $1\ \lceil \frac{1}{2}n \rceil\ \lceil \frac{1}{2}n \rceil - 1\ \ldots\ 2$).

(ii) Note that stage 1* only requires 2n messages and leaves at most $\frac{1}{2}n$ processors (peaks) that will send a message on the ring at the end of the stage. To allow for an analysis based on random sequences, we note that this is only an optimized version of the algorithm in which every processor X_i sends a message on the ring in a direction as determined in stage 1*. By pairing every permutation with one in which the neighbors of X_i are interchanged, one easily sees that X_i sends its messages to the left or to the right with probability $\frac{1}{2}$ (averaged over all permutations). The message sent by X_i will be annihilated by the first upper record X_j in the direction determined in stage 1*, or by the message of the first upper record that is a peak in the same direction (in case this message was sent towards X_i and collided with the $<X_i>$-message between X_i and X_j). We ignore the case that X_i does not have an upper record. Without loss of generality we may in fact assume that X_i and X_j are more than two steps apart, otherwise the $<X_i>$-message certainly travels only O(1) steps. As a result we may assume that X_j sends a message towards X_i with probability $\frac{1}{2}$, where we note that the complementary case with probability $\frac{1}{2}$ consists of X_j sending its message away from X_i or not sending a message at all. (This is seen by the following argument, where we use X_j^l and X_j^r to denote the left and right neighbors of X_j as seen from X_i in the direction of X_j. Note that $X_j^l < X_i$, by the assumption that X_j is the first upper record. If $X_j^r < X_i$ then pair the current permutation with the one in which X_j^l and X_j^r are switched. If $X_j^r >$ X_i then pair the current permutation with the one in which X_j and X_j^r are switched. Of every pair precisely one permutation will give a case in which the first upper record of X_i sends a message towards X_i. Note that the pairing of permutations is independent of the choice of a direction by X_i.) By theorem 2.2.1.2. we know that the average distance of a random processor to its first upper record is H_n. It follows that Algorithm-D uses (at most) $\frac{3}{4}nH_n+O(n)$ messages on the average. (One should observe that, as in the proof of theorem 2.2.2.1. (ii), the analysis ignores the possible effect of higher order upper records. Thus the average number of messages used by Algorithm-D will actually be less than the claimed $\frac{3}{4}H_n+O(1)$ messages per processor, cf. section 2.2.3.) □

Corollary 2.2.2.3. Decentralized extrema-finding can be achieved strictly more efficiently (i.e., with fewer messages on the average) for bidirectional rings than for unidirectional rings.

Note that Algorithm-P and Algorithm-D use "time" n and n+1, respectively, when executed under ideal assumptions, not counting the time for the inauguration of an elected leader.

2.2.3. An improved analysis of Algorithm-P and Algorithm-D.

In the proofs of theorem 2.2.2.1. and 2.2.2.2. it was argued that the bound of $\frac{3}{4}H_n$ on the average (c.q. expected) number of propagations of an $<X_i>$-message is only an upperbound, because the possible effect of higher order upper records was ignored. In this section we will improve the analyses and derive a bound of about $0.7H_n$ on the average (c.q. expected) number of propagations of a message in both algorithms.

For an analysis of Algorithm-P, we assume without loss of generality that i=1 and that the $<X_1>$-message is sent to the right. Let v_1, v_2, ... be random variables denoting the position of the first and higher order upper records (cf. section 2.2.1). If X_{v_1} to $X_{v_{j-1}}$ randomly choose to send their <X>-message to the right as well but X_{v_j} sends its message to the left, then the $<X_1>$-message is annihilated by the $<X_{v_j}>$-message if the messages meet before X_{v_1} is reached, i.e., at processor $X_{1+\frac{\lceil v_j-1 \rceil}{2}}$ provided $v_j<2v_1$. (For v_j-1 odd, the messages will not meet but pass over the same link before $<X_1>$ is annihilated at the next processor.) Otherwise the $<X_1>$-message is simply killed at X_{v_1}.

Definition. $K_n(j)= \sum\limits_{\substack{1<t_1<..<t_j\leq n \\ t_j<2t_1}} \dfrac{t_1-\frac{1}{2}t_j}{(t_1-1)\ldots(t_j-1)t_j}$.

Suppose that we take the effect of up to l upper records into account. (Further upper records could only lower the bound on the expected message complexity.)

Theorem 2.2.3.1. The expected number of messages used by Algorithm-P is bounded by $n(H_n - \sum\limits_{j=1}^{l} (\frac{1}{2})^j K_n(j)) + O(n)$.

Proof.

The expected number of $<X_1>$-messages propagated by Algorithm-P is bounded by the expected value of

$$\sum_{j=1}^{l} (\tfrac{1}{2})^j \frac{\lceil v_j-1 \rceil}{2} + (\tfrac{1}{2})^l (v_1-1) =$$

$$= (v_1-1)-\left\{ \sum_{j=1}^{l} (\tfrac{1}{2})^j (v_1-1)- \sum_{j=1}^{l} (\tfrac{1}{2})^j \frac{\lceil v_j-1 \rceil}{2} \right\}$$

$$= (v_1-1)- \sum_{j=1}^{l} (\tfrac{1}{2})^j \left[v_1-\frac{v_j}{2} \right] +O(1)$$

, where each term in the summation arises with the probability of v_j being less than $2v_1$ (and thus of the $<X_{v_j}>$-message serving as the annihilator of the $<X_1>$-message). We ignore the effect of rings without a first upper record. The expected value is given by

$$\sum_{1<t_1\le n} P(v_1=t_1)(t_1-1)-\sum_{j=1}^{1}(\tfrac{1}{2})^j \sum_{\substack{1<t_1<\ldots<t_j\le n\\ t_j<2t_1}} P(v_1=t_1;\ldots;v_j=t_j)(t_1-\tfrac{1}{2}t_j)+O(1)=$$

$$=\sum_{1<t_1\le n}\frac{t_1-1}{(t_1-1)t_1}-\sum_{j=1}^{1}(\tfrac{1}{2})^j K_n(j)+O(1)=$$

$$=H_n-\sum_{j=1}^{1}(\tfrac{1}{2})^j K_n(j)+O(1),$$

using theorem 2.2.1.3. Accumulating this bound for all $<X_i>$-messages yields the result. □

Note that the term $-(\frac{1}{2})^j K_n(j)\cdot n$ denotes the savings in the expected number of messages used by the algorithm when the effect of the j^{th} upper record is taken into account.

Lemma 2.2.3.2.

(i) $K_n(1) = \frac{1}{2}H_n + O(1)$,

(ii) $K_n(2) = (\frac{1}{2} - \frac{1}{2}\ln 2)\ H_n + O(1)$,

(iii) $K_n(3) = (\frac{1}{2} - \frac{1}{2}\ln 2 - \frac{1}{4}\ln^2 2)\ H_n + O(1)$,

(iv) $K_n(j+1) < K_n(j)$.

Proof.

(i) $K_n(1) = \sum_{1<t_1\le n}\frac{\frac{1}{2}}{(t_1-1)} = \frac{1}{2}H_n + O(1)$.

(ii) See appendix A.

(iii) See appendix A.

(iv) The result is obvious by the interpretation of $K_n(j+1)$ and $K_n(j)$, but can be proved formally as follows.

$$K_n(j+1)=\sum_{\substack{1<t_1<\ldots<t_{j+1}\le n\\ t_{j+1}<2t_1}}\frac{t_1-\frac{1}{2}t_{j+1}}{(t_1-1)\ldots(t_{j+1}-1)t_{j+1}}=$$

$$= \sum_{1<t_1<\ldots<t_j\le n} \left\{ \frac{1}{(t_1-1)\ldots(t_j-1)} \sum_{\substack{t_j<t_{j+1}\le n\\ t_{j+1}<2t_1}} \frac{t_1-\frac{1}{2}t_{j+1}}{(t_{j+1}-1)t_{j+1}} \right\} \le$$

$$\le \sum_{\substack{1<t_1<\ldots<t_j\le n\\ t_j<2t_1}} \left\{ \frac{1}{(t_1-1)\ldots(t_j-1)t_j} \sum_{t_{j+1}=t_j+1}^{2t_1-1} \frac{t_j\cdot(t_1-\frac{1}{2}t_{j+1})}{(t_{j+1}-1)t_{j+1}} \right\} <$$

$$< \sum_{\substack{1<t_1<\ldots<t_j\le n\\ t_j<2t_1}} \left\{ \frac{t_1-\frac{1}{2}t_j}{(t_1-1)\ldots(t_j-1)t_j} \sum_{t=t_{j+1}}^{2t_1-1} \frac{t_j}{(t-1)t} \right\} <$$

$$< K_n(j),$$

where we use that $\sum_{t=t_j+1}^{2t_1-1} \frac{t_j}{(t-1)t}$ consists of at most t_1-2 terms of size less than $\frac{1}{t_1}$ and thus has a sum <1 (but positive). □

Theorem 2.2.3.3. The expected number of messages used by Algorithm-P is equal to $0.70\ldots nH_n + O(n)$.

Proof.

By theorem 2.2.3.1. the expected number of messages used by Algorithm-P is equal to $n(H_n - \sum_{j=1}^{L} (\frac{1}{2})^j K_n(j)) + O(n)$, for the largest L possible. (Note that no ring has a j^{th} upper record with $t_j<2t_1$ for $j\ge \frac{1}{2}n+1$.) Using lemma 2.2.3.2. this number is bounded from above by

$$(1 -\tfrac{1}{2}(\tfrac{1}{2}) -\tfrac{1}{4}(\tfrac{1}{2}-\tfrac{1}{2}\ln 2) -\tfrac{1}{8}(\tfrac{1}{2}-\tfrac{1}{2}\ln 2-\tfrac{1}{4}\ln^2 2))nH_n + O(n) =$$
$$= (\tfrac{9}{16} + \tfrac{3}{16}\ln 2 + \tfrac{1}{32}\ln^2 2)nH_n + O(n) =$$
$$= 0.7075nH_n + O(n),$$

and from below by

$$\left(\frac{9}{16} + \frac{3}{16}\ln 2 + \frac{1}{32}\ln^2 2\right) nH_n - \left(\frac{1}{2}-\frac{1}{2}\ln 2-\frac{1}{4}\ln^2 2\right) \sum_{j=4}^{\infty} \left(\frac{1}{2}\right)^j \cdot nH_n + O(n)$$
$$= \left(\frac{1}{2} + \frac{1}{4}\ln 2 + \frac{1}{16}\ln^2 2\right) nH_n + O(n) =$$
$$= 0.7033\ nH_n + O(n).\quad \square$$

Because of the analogy to Algorithm-P (cf. theorem 2.2.2.2), it is intuitive that the improved bound of $0.70..nH_n + O(n)$ messages also holds for the average number of messages used by Algorithm-D. We give a more rigorous proof of this fact. Note that in its first stage, Algorithm-D expends $O(n)$ messages to eliminate every processor that has a larger neighbor. The active processors that remain are at least one position apart, but must be at least two positions apart if we want to claim the independence of their choice of direction at the end of stage 1* (cf. theorem 2.2.2.2.). This motivates the definition of a modified record concept.

Definition. Given a random sequence, and "upper *-record" is any element that is larger than all the preceding ones (thus an upper record in the traditional sense) for which the two immediately preceding elements are not upper *-records.

We will bound the average number of messages used by Algorithm-D by taking only the effect of upper *-records into account, by an analysis that is very similar to the analysis of Algorithm-P. (The effect of upper records that are no upper *-records will only lower the resulting estimate.)

Definition. $L_n(j) = \Sigma\, P^*(v_1=t_1;\ldots;v_j=t_j)\cdot(t_1-\frac{1}{2}t_j)$, where the summation is taken over all $t_1,\ldots,t_j$ with $1<t_1<\ldots<t_j\leq n$ and $2<t_1$, $t_1+2<t_2,\ldots,t_{j-1}+2<t_j$ and $t_j<2t_1$.

In the definition, $P^*(v_1=t_1;\ldots;v_j=t_j)$ denotes the probability that the i^{th} upper *-record occurs at position t_i $(1i\leq j)$. Note that for $2<t_1$, $t_1+2<t_2,\ldots,t_{j-1}+2<t_j$ one has $P^*(v_1=t_1;\ldots;v_j=t_j) \geq P(v_1=t_1;\ldots;v_j=t_j) = \frac{1}{(t_1-1)\ldots(t_j-1)t_j}$, because the upper records in positions t_1 through t_j will automatically be upper *-records. Suppose

we only take the effect of 1 upper *-records into account. In analogy to theorem 2.2.3.1. one obtains the following fact.

Theorem 2.2.3.4. The average number of messages used by Algorithm-D is bounded by (at most) $n(H_n - \sum_{j=1}^{1} (\frac{1}{2})^j L_n(j)) + O(n)$.

For simplicity define $L'_n(j) = \Sigma \frac{t_1 - \frac{1}{2}t_j}{t_1-1)\dots(t_j-1)t_j}$, where the summation extends over all $t_1,\dots,t_j$ as in $L_n(j)$.

Lemma 2.2.3.5.

(i) $L_n(j) \geq L'_n(j)$,

(ii) $0 \leq K_n(j) - L'_n(j) = O(j)$.

Proof.

(i) This follows immediately from the definitions.

(ii) Obviously $K'_n(j) \geq L'_n(j)$, as L'_n equals the expression for $K_n(j)$ over a more restricted range. To show that $K_n(j) - L'_n(j) = O(j)$ we define the following auxiliary quantity $M_n(j,i)$ for $0 \leq i \leq j$:

$$M_n(j,i) = \Sigma \frac{t_1 - \frac{1}{2}t_j}{(t_1-1)\dots(t_j-1)t_j},$$ where the summation is taken over all $t_1,\dots,t_j$ with $1<t_1<\dots<t_j \leq n$ and $t_i+2<t_{i+1},\dots,t_{j-1}+2<t_j$ and $t_j<2t_1$.

Observe that $K_n(j) = M_n(j,j)$ and $L'_n(j) = M_n(j,0)$, and clearly $M_n(j,i+1) \geq M_n(j,i)$. Note that $\frac{t_1 - \frac{1}{2}t_j}{t_j} \leq 1$, and that for "fixed" t_1 to t_i one has $\Sigma \frac{t_1 - \frac{1}{2}t_j}{(t_1-1)\dots(t_j-1)(t_{i+1}-1)\dots(t_j-1)t_j} = O\left[\frac{1}{(t_1-1)\dots(t_i-1)}\right]$, where the summation extends over all $1<t_{i+1}<\dots<t_j \leq n$ with $t_i+2<t_{i+1},\dots,t_{j-1}+2<t_j$ and $t_j<2t_1$. Next observe that

$$M_n(j,i+1) - M(j,i) = \Sigma \frac{t_1 - \frac{1}{2}t_j}{(t_1-1)\dots(t_j-1)t_j}$$

where the summation is taken over all $t_1,\dots,t_j$ with $1<t_1<\dots<t_j \leq n$ and $t_{i+1}+2<t_{i+2},\dots,t_{j-1}+2<t_j$ and $t_j<2t_1$, and $t_{i+1} = t_i+1$ or $t_{i+1} = t_i+2$. It follows that

$$M_n(j,i+1)-M(j,i) = O\left(\sum_{\substack{1<t_1<\dots<t_{i+1}\le n\\ t_{i+1}<2t_1\\ t_{i+1}=t_i+1 \text{ or } =t_i+2}} \frac{1}{(t_1-1)\dots(t_i-1)(t_{i+1}-1)}\right)=$$

$$= O\left(\sum_{\substack{(1<t_1<..<t_i\le n)\\ t_i<2t_1}} \frac{1}{(t_1-1)..(t_i-1)t_i}\right)=$$

$$= O\left(\sum_{1<t_1\le n} \frac{1}{t_1^2}\right) = O(1).$$

Hence $K_n'(j)-L_n(j) = M_n(j,j)-M_n(j,o) = \sum_{i=0}^{j-1} (M(j,i+1)-M(j,i)) = O(j)$. □

Theorem 2.2.3.6. The average number of messages used by Algorithm-D is bounded by (at most) $0.7075nH_n + O(n)$.

Proof.

By theorem 2.2.3.4. and lemma 2.2.3.5. the average number of messages used by Algorithm-D can be bounded as follows (for any fixed l):

$$n(H_n-\sum_{j=1}^{l}(\tfrac{1}{2})^j L_n(j))+O(n) \le n(H_n-\sum_{j=1}^{l}(\tfrac{1}{2})^j L_n'(j))+O(n) =$$

$$= n(H_n-\sum_{j=1}^{l}(\tfrac{1}{2})^j K_n(j))+O(n\cdot\sum_{j=1}^{l}(\tfrac{1}{2})^j j)+O(n) =$$

$$= n(H_n-\sum_{j=1}^{l}(\tfrac{1}{2})^j K_n(j))+O(n).$$

The result now follows by using the upper bound on the latter expression, derived in the proof of theorem 2.2.3.3. □

Recently, this analysis was improved by Flayolet [Fl86], who obtained an exact bound of $\frac{\sqrt{2}}{2}\cdot nH_n + O(n)$ for the average number of messages used by Algorithm-P. With the techniques, exposed in this section one can show that this bound also is an upperbound for the average number of messages used by Algorithm-D. We conjecture that the average number of messages used by Algorithm-D also is exactly $\frac{\sqrt{2}}{2}\cdot nH_n + O(n)$.

2.3. Some lowerbound results on decentralized extrema finding in a ring of processors.

2.3.1. Introduction. In this section we will derive a number of lowerbound results for the decentralized extrema finding problem in rings of processors. The first lowerbound for this problem was obtained by Burns [Bu80], who proved a lowerbound of $\frac{1}{4}$ n log n messages for the worst-case for bidirectional rings. Pachl, Korach and Rotem [PKR82] proved a lowerbound of $nH_n \approx 0,69$ n log n on the average number of messages required by a unidirectional algorithm (hence the Chang-Roberts algorithm is optimal with respect to the average number of messages) and a lowerbound of $\frac{1}{8}$ n log n + O(n) on the average number of messages for bidirectional rings. For the case that the size of the ring n is initially known to the processors, they proved lowerbounds of $\frac{6}{5\log 5}$ n log n + O(n) and $\frac{1}{4}$n log n + O(n) on the worst-case number of messages, for unidirectional rings and for bidirectional rings, respectively. (No non-trivial lowerbounds are known on the average number of messages needed for rings with known ring size.)

For the synchronous case, Frederickson and Lynch [FL84] prove a lowerbound of $\frac{1}{2}$n log n + O(n) on the worst-case number of messages for (bidirectional) comparison algorithms (a comparison algorithm uses only mutual comparisons between identification numbers (=, ≠, <, >, ≤, ≥)), and for (bidirectional) algorithms, that run in time, bounded by some constant t_n on all rings with size n. This lowerbound is also valid for the asynchronous case: every algorithm that runs on a asynchronous ring can also run on a synchronous ring, and the time the algorithm uses is bounded, for instance by the number of messages that is sent.

This section is organized as follows. In section 2.3.2. we give some preliminary definitions and results. In section 2.3.3. we relate the worst-case number of messages for arbitrary (asynchronous) algorithms to this number for (asynchronous) comparison algorithms. Our results and proof-techniques we use in this section are very similar to results, proven by Frederickson and Lynch [FL84] for the synchronous case. The results of section 2.3.3. will be used in section 2.3.4. to answer a question, posed by Korach, Rotem and Santoro in 1981 [KRS81], whether algorithms that use time n, have to send a quadratic number of messages. The answer will be positive or negative, depending on the precise assumptions made about whether we want to find the maximum or whether we want to find a leader (which does not have to be the maximum) and what processors have to "know" the maximum or leader after completion of the algorithm. In figure 2.3.1.1 the results of section 4 are summarized. In section 2.3.5. we consider the case where the size of the ring n is initially known to the processors, for unidirectional rings. We give a better lowerbound on the worst-case number of messages and we give a lowerbound on the average number of messages for comparison algorithms.

2.3.2. Definitions and preliminary results. Pachl, Korach and Rotem [PKR82] introduced the notion of full-information algorithms (for unidirectional rings). In full-information algorithms, when a processor sends a message, it sends everything it knows. In this way, algorithms have to specify only when to send (and no longer what to send). Every algorithm A corresponds to an 'equivalent' full-information algorithm A' : if during execution of algorithm A', a processor p receives a message s from a neighboring processor, p can decide from its own knowledge, the information that is contained in the message s, and the direction from which it received s, whether or not it would have sent a message on the corresponding moment during execution of algorithm A. A and A' use the same number of messages (although messages sent by A' can be considerably longer than messages sent by A) and the same time. (We

Problem	Unidirectional	Bidirectional
-every processor must know the maximum	$\geq \frac{1}{2} n\ (n+1)$	$\Omega\ (n^2)$
-one or more arbitrary processors must know the maximum	$\leq 1.356\ n \log n + O(n)$	$\leq 1.356\ n \log n + O(n)$
-the maximum must know it is the maximum	$\geq \frac{1}{2} n(n+1)$	$\Omega(n^2)$
-one processor must declare itself as a leader	$\leq 1.356\ n \log n + O(n)$	$\leq 1.356\ n \log n + O(n)$
-one processor must declare itself as a leader, and all processors must know the id. of the leader	$\geq \frac{1}{2} n\ (n+1)$	$\Omega(n^2)$

Fig. 2.3.1.1. General bounds for problems that have to be solved in time $\leq$ n.

assume that in one time unit each processor can send one message. We ignore the time used for calculations in processors.) We also assume that identification numbers are chosen from Z.

We will first consider unidirectional full-information algorithms. In a unidirectional full-information algorithm, the 'knowledge' of a processor consists of its identification number (abbreviated: its id) and the id's of a number of processors (initially zero) 'before' it on the ring. If a processors with identification number id_* receives a message $<id_1 \ldots id_k>$, then either it sends a message $<id_1 \ldots id_k\ id_*>$ to the next processor, or it does nothing. If a processor id_* receives a message $<id_1...id_n>$ and $id_* = id_1$, then it knows the id of every processor on the ring. It depends on what exactly we want the algorithm to do whether we have to send some more messages. If we want an arbitrary processor to know the maximum, then we are done. If we want that the processor whose id is the maximum to become aware of this fact, or if we want every processor to know the maximum, then some extra messages may have to be sent. In both cases, it is not difficult to find the most efficient way to finish the algorithm, and at most n extra messages in total have to be used. We therefore only consider messages with length $\leq n$ = number of processors on the ring.

Definitions.

(i) Let $X \subseteq Z$. D(X) is the set of finite, non-empty sequences of distinct elements of X, i.e.
$D(X) = \{<s_1...s_k>| k\geq 1,\ 1\leq i\leq k \Rightarrow s_i \in X;\ i\neq j \Rightarrow s_i \neq s_j\}$.

(ii) D = D(Z).

(iii) Let $s\in D$. len(s) is the length of s, i.e. $len(<s_1...s_k>) = k$.

(iv) Let $s\in D$. C(s) is the set of cyclic permutations of s.

(v) Let $s,t \in D$. t is a subsequence of s, if there are $u, v \in D \cup \{\varepsilon\}$ (ε is the empty sequence), with s=utv.

(vi) For $s\in D$, $E \subseteq D$ let $N(s,E) = |\{t\in E|$ t is a subsequence of an element of $C(s)\}|$.

Definition. Let $E \subseteq D$. E is exhaustive, iff

(i) $\forall\ t,u\in D : tu\in E \Rightarrow t\in E$.

(ii) $\forall\ s \in D : C(s) \cap E \neq \emptyset$.

Theorem 2.3.2.1. is a minor extension of a result of Pachl, Korach and Rotem [PKR82].

Theorem 2.3.2.1. [PKR82] Let A be a (unidirectional) full-information algorithm. Let $E = \{t\in D$ | a message with content t will be transmitted when A executes on a ring labeled $t\}$. Then

(i) E is exhaustive

(ii) A requires exactly N(s,E) messages with length $\leq$ len (s), when executed on a ring labeled s (and possibly some extra messages with greater length).

Conversely every effective computable (i.e. recursive) exhaustive set E $\subseteq$ D corresponds to a maximum finding algorithm: use the full information algorithm that sends a message if and only if it is an element of E.

An algorithm is said to be a comparison algorithm iff no other operations on the id's are used as mutual comparisons ($=,\neq,<,>,\leq,\geq$). We now give the corresponding notion for sets.

Definition. Let s, t $\in$ D. $s \equiv t$ (s and t are order equivalent), iff len(s) = len(t), $\forall i,j$ $1 \leq i,j \leq$ len(s), $s_i < s_j \Leftrightarrow t_i < t_j$.

Definition. Let E $\subseteq$ D. E is comparison-based, iff $\forall$ s,t$\in$D: $s \equiv t \Rightarrow (s \in E \Leftrightarrow t \in E)$.

Definition. Let E $\subseteq$ D. E is comparison-exhaustive, iff
(i) E is exhaustive
(ii) E is comparison-based.

Theorem 2.3.2.2. Let A be a (unidirectional) full-information algorithm, corresponding to a comparison algorithm. Then E = {t$\in$D| a message with content t will be transmitted when A executes on a ring labeled t} is comparison-exhaustive.

Again, conversely, effective computable comparison-exhaustive sets E $\subseteq$ D correspond to comparison maximum finding algorithms.

The notion of full information algorithms can be extended to bidirectional algorithms. Our notion of a bidirectional full-information algorithm is very similar to the notion of free algorithms of Frederickson and Lynch [FL84]. (Free algorithms run on bidirectional synchronous rings). In bidirectional algorithms the behavior of a processor does not necessarily depend fully on the id's of the processors in the neighborhood it knows, but can also depend on the order in which it received messages from its neighbors, etc.

Definition. Let $X \subseteq Z$. $D_b(X)$ is the smallest set of strings, such that

(i) $id \in X \Rightarrow \langle id \rangle \in D_b(X)$

(ii) $id \in X, k \geq 1,\ s_1 \ldots s_k \in D_b(X),\ d_1, \ldots, d_k \in \{l, r\} \Rightarrow (\langle id \rangle, \langle (d_1, s_1), (d_2, s_2), \ldots, (d_k, s_k) \rangle) \in D_b(X)$.

We abbreviate $D_b(Z)$ as D_b.

The information that a certain processor has on a certain moment during execution of a bidirectional full-information algorithm can be represented by an element of D_b. For instance, read the string $(\langle id \rangle, \langle (l, s_1), (r, s_2) \rangle)$ as: my own identification number is id, the first message I received came from my left neighbor and contained information s_1; the second and last message I received came from my right neighbor and contained information s_2.

So if a processor sends a message, it sends all its information, that is: an element of D_b. If a processor p with information $(\langle id \rangle, \langle (d_1, s_1), .., (d_k, s_k) \rangle)$ receives a message s, its new information becomes $(\langle id \rangle, \langle (d_1, s_1), .., (d_k, s_k), (d_{k+1}, s) \rangle)$, with $d_{k+1} = l$ if the message came from p's left neighbor and $d_{k+1} = r$, if the message came from p's right neighbor.

Now every bidirectional full-information algorithm corresponds to a pair (L,R), L,R subsets of D_b. L corresponds to the messages that are sent to left neighbors, R corresponds to the messages that are sent to right neighbors, in a manner similar to the undirected case. Note that a message that is sent to both neighbors will be a member of $L \cap R$. We will not give a bidirectional variant of the notion of exhaustiveness.

Definition. Let $s \in D_b$. $\tilde{s}$ is obtained by taking the successive integers that appear in s, ignoring other characters:

- if s is of the form $\langle id \rangle$, with $id \in Z$, then $\tilde{s} = id$
- if s is of the form $\langle id, \langle (d_1, s_1), (d_2, s_2), \ldots, (d_k, s_k) \rangle \rangle$, then $\tilde{s} = id \circ \tilde{s}_1 \circ \tilde{s}_2 \circ \ldots \circ \tilde{s}_k$, (where $a_1 \ldots a_n \circ b_1 \ldots b_m = a_1 .. a_n b_1 .. b_m$).

Definition. Let $s,t \in D_b$. $s \equiv t$ (s and t are order equivalent), iff $len(\tilde{s}) = len(\tilde{t})$ and $\tilde{s}_i < \tilde{s}_j \Leftrightarrow \tilde{t}_i < \tilde{t}_j$ for all i,j, $1 \le i,j \le len(\tilde{s})$. ($\tilde{s}_i$ is the i'th integer in the string $\tilde{s}$.)

Definition. Let $s,t \in D_b$. $s \Box t$ (s and t are equally typed), iff

a) s and t are both of the form <id>, with $id \in Z$,

or

b) there exists a $k \in Z$ and $d_1,\ldots,d_k \in \{l,r\}$, such that

(i) s is of the form $<id_s,<(d_1,s_1),\ldots,(d_k,s_k)>>$;

(ii) t is of the form $<id_t,<(d_1,s_1),\ldots,(d_k,s_k)>>$;

(iii) For all i, $1 \le i \le k$, $s_i \Box t_i$.

We call a set $E \subseteq D_b$ comparison based, iff for all s,t with $s \Box t$ and $s \equiv t : s \in E \Leftrightarrow t \in E$. Similar to the undirected case one has:

Theorem 2.3.2.3. Let A be a bidirectional full-information comparison algorithm. Let L, $R \subseteq D_b$ be the sets of all messages that could possibly be sent by algorithm A to left and right neighbors, respectively. Then L and R are comparison based sets.

2.3.3. Arbitrary versus comparison algorithms. In this section we relate the worst-case number of messages needed in arbitrary and comparison algorithms. The results and proof-techniques are very similar to results, proven by Frederickson and Lynch [FL84] for the synchronous case. In the proof we will use a well-known Ramsey theorem (theorem 2.3.3.1.). We first consider the unidirectional case.

Theorem 2.3.3.1. (Ramsey's theorem, see e.g. [Ba77].) Let A be an infinite set, $k,n \in N^+$, and $C_1,..,C_k$ a partition of $P^n(A)$. ($\bigcup_{i=1}^{k} C_i = P^n(A)$, $i \ne j \Rightarrow C_i \cap C_j = \emptyset$). Then there exists a homogeneous subset $B \subseteq A$, i.e. $\exists i$ $P^n(B) \subseteq C_i$.

Definitions.

(i) Let $E \subseteq D$ be exhaustive, and let $A \subseteq Z$, $n \in N^+$. We let $E(n,A)$ denote the set of all strings in E with elements in A and length at most n, i.e. $E(n,A)=\{s \in E \subseteq D(A) \mid \mathrm{len}(s) \leq n\}$.

(ii) Let $n \in N^+$. We fix an enumeration of all permutations of the set $\{1,..,n\}$, denoted by $\pi_1^n, \pi_2^n, \ldots, \pi_{n!}^n$.

(iii) Let $X \subseteq N$, $n=|X|$, and $1 \leq i \leq n!$. We let $\pi_i^n(X)$ denote the string, obtained by placing the elements of X, in the order, prescribed by π_i^n, i.e. $\pi_i^n(X) \in D(X)$; $\pi_i^n(X) \equiv \pi_i^n$.

Theorem 2.3.3.2. Let $E \subseteq D$ be exhaustive.
There exists a collection of infinite sets $A_1, A_2, \ldots, A_n, \ldots$, with

(i) $A_1 = Z$

(ii) $\forall n \in N^+$; $A_n \supseteq A_{n+1}$

(iii) $\forall n \in N^+$: $E(n,A_n)$ is comparison based.

Proof. We use induction to n. First note that E contains every string $<v>$, $v \in Z$. (We use that E is exhaustive). So $E(1,Z) = E(1,A_1)$ is comparison based.

Now let an infinite set A_n be given, with $E(n,A_n)$ comparison based. We will now show that A_{n+1} can be chosen, such that it fulfills the requirements.

With induction we define a row of infinite sets $B_o^{n+1}, \ldots, B_{(n+1)!}^{n+1}$, as follows:

- $B_o^{n+1} = A_n$.
- Let with induction an infinite set B_j^{n+1} be given. Let $C_1=\{X \subseteq P^{n+1}(B_j^{n+1}) \mid \pi_{j+1}^{n+1}(X) \in E\}$ and $C_2=\{X \subseteq P^{n+1}(B_j^{n+1}) \mid \pi_{j+1}^{n+1}(X) \notin E\}$. Ramsey theorem 2.3.3.1. with k=2 gives that we can choose B_{j+1}^{n+1} such that:

(i) $B_{j+1}^{n+1} \subseteq B_j^{n+1}$

(ii) B_{j+1}^{n+1} is infinite

(iii) $P^{n+1}(B_{j+1}^{n+1}) \subseteq C_1$ or $P^{n+1}(B_{j+1}^{n+1}) \subseteq C_2$.

Finally choose $A_{n+1}=B_{(n+1)!}^{n+1}$. It is obvious that A_{n+1} is infinite and $A_n \supseteq A_{n+1}$.

Now suppose s_1, $s_2 \in D(A_{n+1})$ and $len(s_1) = len(s_2) \leq n+1$ and $s_1 \equiv s_2$. If $len(s_1) \leq n$, then $s_1 \in E \Leftrightarrow s_2 \in E$, because $A_n \supseteq A_{n+1}$ and $E(n,A_n)$ is comparison based. Now suppose $len(s_1) = n+1$. There exists a unique π^{n+1}_k, with $s_1 \equiv \pi^{n+1}_k \equiv s_2$. Let S_1, S_2 be the sets of integers, appearing in s_1, s_2 respectively. $(s_i = \langle id_1 \ldots id_n \rangle \Rightarrow S_i = \{id_i, \ldots, id_n\})$. $S_1, S_2 \subseteq B^{n+1}_k$. The construction of B^{n+1}_k shows that $\pi^{n+1}_k(S_1) = s_1 \in E \Leftrightarrow S_1 \in C_1 \Leftrightarrow P^{n+1}(B^{n+1}_k) \subseteq C_1 \Leftrightarrow S_2 \in C_2 \Leftrightarrow \pi^{n+1}_k(S_2) = s_2 \in E$. So again $s_1 \in E \Leftrightarrow s_2 \in E$. This shows that $E(n+1,A_{n+1})$ is comparison based. □

Theorem 2.3.3.3. Let $E \subseteq D$ be exhaustive. Then there exists a comparison-exhaustive set $F \subseteq D$, such that

$$\forall n \max\{N(s,F) \mid s \in D,\ len(s)=n\} \leq \max\{N(s,E) \mid s \in D,\ len(s)=n\}.$$

Proof. Let an exhaustive set $E \subseteq D$ be given, and let the sets $A_1, A_2, \ldots, A_i \ldots$, as implied by theorem 2.3.3.2., be given. Now choose $F = \{S \in D \mid \exists\, t \in E \cap D(A_{len(s)}) : t \equiv s\}$. F is comparison-exhaustive:

(i) Let $t,u \in D$ and $tu \in F$. Then $\exists\ sv \in E \cap D(A_{len(tu)}) :\ sv \equiv tu \wedge len(s)=len(t)$. $\Rightarrow \exists\ s \in E \cap D(A_{len(s)}) : s \equiv t \Rightarrow t \in F$.

(ii) Let $s \in D$. One can choose $v \in D(A_{len(s)})$, with $v \equiv s$. (Choose len(s) different elements from $A_{len(s)}$ and place them in the right order.) Now $C(v) \cap E \neq \emptyset$. Let $w \in C(v) \cap E$. We can find a $t \in C(s)$, with $t \equiv w$. (Use the same cyclic permutation that is necessary to obtain w from v to obtain t from s.)

So $t \equiv w$ and $w \in D(A_{len(t)}) \cap E$. This implies $t \in F$ and $C(s) \cap F \neq \emptyset$.

(iii) Let $s,t \in D$ and $s \equiv t$. Then

$$\begin{aligned} s \in F &\Leftrightarrow \exists\ v \quad v \equiv s \cap v \in D\ (A_{len(s)}) \\ &\Leftrightarrow \exists\ v \quad v \equiv t \cap v \in D\ (A_{len(t)}) \\ &\Leftrightarrow t \in F. \end{aligned}$$

This concludes the proof that F is comparison-exhaustive.

Finally we will show that for all n:

$$\max\ \{N(s,F) \mid s \in D,\ len(s) = n\} \leq \max\ \{N(s,E) \mid s \in D,\ len(s) = n\}.$$

Let n be given. Let $s \in D$, with $len(s)=n$. There exists a $t \in D(A_n)$, with

$t \equiv s$. It follows directly from the exhaustiveness of F that $N(s,F) = N(t,F)$. Further, for every subsequence t_o of t: $t_o \in E \Leftrightarrow t_o \in F$ (use that $t_o \in D(A_n) \subseteq D(A_{len(t_o)})$), hence $N(t,F) = N(t,E)$. So for all $s \in D$, $len(s)=n$ there exists a $t \in D$, $len(t)=N$, with $N(s,F) = N(t,E)$. So

$$\max \{N(s,F) \mid s \in D, len(s)=n\} \leq \max \{N(s,E) \mid s \in D, len(s)=n\}. \quad \square$$

An algorithmic variant of theorem 2.3.3.2 is the following:

Theorem 2.3.3.4. Let A be a (unidirectional) extrema finding algorithm for rings. There exist algorithms $B_1, B_2, \ldots, B_i, \ldots$ with for every $i \in N^+$:

(i) algorithm B_i works correctly on every ring with size $\leq i$.

(ii) algorithm B_{i+1} is an extension of algorithm B_i (i.e. algorithm B_{i+1} sends the first i steps exactly the same messages as algorithm B_i.)

(iii) B_i is a comparison algorithm.

(iv) For every $j \leq i$, the worst-case number of messages used by B_i on rings with size j is lesser or equal than the worst- case number of messages used by algorithm A on rings with size j.

Proof. Let A' be the full information variant of algorithm A. Let $E = \{t \in D \mid$ a message with content t will be transmitted when A' executes on a ring labeled t$\}$. E is exhaustive (cf. theorem 2.3.2.1.). Let the sets $A_1, A_2, \ldots, A_i, \ldots$ be given, as implied by theorem 2.3.3.2. Now let algorithm B_i be the (full information) algorithm, that sends a message s, if and only if there is a $t \equiv s$, with $t \in D(A_i) \cap E$. We first have to show that B_i indeed is a comparison algorithm, i.e. that it is computable using only comparisons whether there exists a $t \equiv s$ with $t \in D(A_i) \cap E$. Note that if a certain message s is sent by B_i, then also all messages t, with $t \equiv s$ are sent. B_i can use a list of all permutations of i or fewer elements, and with each permutation π_k^j it is stored whether messages s, with $s \equiv \pi_k^j$ are to be sent or not. For each s, one can find, using comparisons only, for which π_k^j $s \equiv \pi_k^j$, and then one can look up whether to send s or not.

It is not difficult to check that (i) - (iv) hold. (Compare the proof of theorem 2.3.3.2.) □

Unfortunately, theorem 2.3.3.3. does *not* imply the following conjecture:

Conjecture. Let A be a (unidirectional) extrema finding algorithm for rings. There exists a (unidirectional) comparison algorithm B (for the extrema finding problem for rings) such that for every n, the worst-case number of messages used by algorithm B on rings with size n is at most the worst-case number of messages used by algorithm A on rings with size n.

The key difference between this conjecture and theorem 2.3.3.3. is the computability of F. To let F "yield an algorithm", it must be effectively computable (i.e. recursive) whether a given $s \in D$ is an element of F or not. This is not necessarily true. However, the results of this section show that some worst-case lowerbound proofs for comparison-algorithms are also valid for arbitrary algorithms. This is when the fact that the comparison algorithm is really an *algorithm* (that is: effectively computable) is not used. For instance, a proof can show a lowerbound for $\max\{N(s,F) \mid s \in D, \text{len}(s)=n\}$ for all comparison-exhaustive sets, not only for *recursive* comparison-exhaustive sets. The results of this section show that this not only implies the same lowerbound on the worst-case number of messages, that are sent on rings with size n by comparison algorithms, but also by arbitrary algorithms.

The results can be generalized to the bidirectional case. A bidirectional variant of theorem 2.3.3.2. is:

Theorem 2.3.3.5. Let A be a bidirectional full-information algorithm, and let L and R be the sets of all messages that could possibly be sent by algorithm A to left and right neighbors, respectively. There exists a collection of infinite sets $A_1, A_2, \ldots,$ with

(i) $A_1 \subseteq Z$

(ii) for all $n \in N^+$: $A_n \supseteq A_{n+1}$

(iii) for all $n \in N^+$ and all strings $s,t \in D_b(A_n)$, with $s \square t$, $\tilde{s} \equiv \tilde{t}$ and

$\text{len}(\tilde{s}) = \text{len}(\tilde{t}) \le n$ one has $s \in L \Leftrightarrow t \in L$ and $s \in R \Leftrightarrow t \in R$.

Proof. The proof is more or less similar to the proof of theorem 2.3.3.2. We will only stress the differences. As in the proof of theorem 2.3.3.2. we use induction to n. Where we had to use one "Ramsey-step" for each permutation of n elements in the unidirectional case, (i.e. for each equivalence class of the strings in D, induced by the equivalence relation $\equiv$), here we have to use the Ramsey theorem for every combination of a type of strings s, with $\text{len}(\tilde{s})=n$, together with an ordering of n elements, i.e. for each equivalence class of the strings in D_b, induced by the equivalence relation $\tilde{=}$, where $\tilde{=}$ is defined by:

$s \tilde{=} t \quad \Leftrightarrow \quad (s \square t \wedge s \equiv t)$.

In every "Ramsey-step" we divide the subsets of the current B_i^{n+1} with the right cardinality in four classes: those that correspond with messages sent to the left and to the right neighbor, those that correspond with messages, sent only to the left neighbor,...etc. Now use theorem 2.3.3.1. with k=4. □

Similar to theorem 2.3.3.4. one now can prove:

Theorem 2.3.3.6. Let A be a (bidirectional) extrema finding algorithm for rings. There exist algorithms $B_1, B_2, \ldots, B_i, \ldots$ with for every $i \in N^+$:

(i) algorithm B_i works correctly on every ring with size $\le i$

(ii) algorithm B_{i+1} is an extension of algorithm B_i (on rings with size $\le i$ algorithm B_i and B_{i+1} behave exactly the same).

(iii) B_i is a comparison algorithm.

(iv) for every $j \le i$, the worst-case number of messages used by B_i on rings with size j, is lesser or equal than the worst-case number of messages used by algorithm A on rings with size j.

Proof. The proof is similar to that of theorem 2.3.3.4. The main difference is that we can no longer use the set A_i to obtain algorithm B_i, because messages on rings with size i can have a size much larger than i. Suppose t_i is an upperbound on the number of timesteps algorithm A takes on any ring with size i. Note that the size of the largest knowledge of any processor can at most triple in one timestep. (That is, when the processor receives equally large messages from both its neighbors). So 3^{t_i} is an upperbound on the size of the largest message ever used by algorithm A on rings with size i, and we can use set $B_{3^{t_i}}$ to obtain algorithm A_i, similar to the unidirectional case. □

The same remarks we made about lowerbound proofs concerning unidirectional rings are also valid for bidirectional rings.

2.3.4. Algorithms that use time ≤ n. In [KRS81] Korach, Rotem and Santoro posed the question "whether algorithms with running time of n must have a quadratic number of messages in their worst-case". The answer to this question depends on what we exactly want the algorithm to do in time n. There are several possibilities:

(A) - every processor must know the maximum (i.e. the id of the largest numbered node)

(B) - there must be one or more arbitrary processors that know the maximum

(C) - the processor that has the largest id must know that he is the node with the largest id

(D) - exactly one processor must declare itself as a leader

(E) - exactly one processor must declare itself as a leader, all other processors must know the id of the leader.

These five problems are closely related. In figure 2.3.4.1. it is shown which problems are subproblems of which other problems. Also, without much difficulty one can show the following relation:

Theorem 2.3.4.1. If there exists an algorithm that solves one of the five problems A-E, and has a worst-case running time t(n), and uses a worst-case number of messages m(n), then for each of the other four problems there exists an algorithm that has a worst-case running time of at most t(n)+2n and uses a worst-case number of messages of at most m(n)+2n.

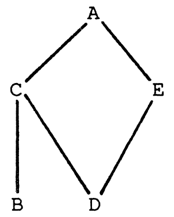


Fig. 2.3.4.1. The partial ordering of the problems A-E. A line indicates that the lower problem is a subproblem of the upper problem.

In figure 2.3.4.2 we summarize the bounds we prove on algorithms that use time ≤n.

Theorem 2.3.4.2. Every unidirectional algorithm for problem C that uses time n, must use at least $\frac{1}{2}n(n+1)$ messages in the worst-case.

Proof. If the processor with the largest id must be aware of the fact that it has the largest id after time n, then it must have sent a message to its successor, and this message must have been propagated around the whole ring, until it returned to the processor, that has originated the message. The exhaustive set, induced by the algorithm must therefore contain all messages $s_1,\ldots,s_n$, with $\forall i\ 2\leq i\leq n\ s_1\geq s_i$. (This means essentially that every message, used by the Chang/Roberts algorithm must be used here too.) Now this implies a worst-case lowerbound of $\frac{1}{2}n(n+1)$ messages on rings with size n. (Consider rings labeled with n, n-1, n-2,...,2,1. Compare this result with the results of Chang and Roberts

problem	unidirectional	bidirectional
A	$\geq\frac{1}{2}n(n+1)$	$\Omega(n^2)$
B	$\leq 1.38\ n \log n+O(n)$	$\leq 1.38\ n \log n+O(n)$
C	$\geq\frac{1}{2}n(n+1)$	$\Omega(n^2)$
D	$\leq 1.38\ n \log n+O(n)$	$\leq 1.38\ n \log n+O(n)$
E	$\geq\frac{1}{2}n(n+1)$	$\Omega(n^2)$

Fig. 2.3.4.2. Bounds for algorithms that use time $\leq n$.

[CR79].) □

Theorem 2.3.4.3. For every bidirectional algorithm for problem C, that uses time $\leq n$, there is a $c \in R^+$, such that the algorithm uses at least cn^2 messages in the worst-case.

Proof. Let A be a bidirectional algorithm for problem C, that uses time $\leq n$, on rings with size n and suppose there does not exist a $c \in R^+$, such that the algorithm uses at least cn^2 messages in the worst-case (on rings with size n). Without loss of generality we can assume that A is a full-information algorithm. We first assume A is a comparison algorithm.

Let rings $r^{k,l}$ be defined as follows: the size of a ring $r^{k,l}$ is $k \cdot l$, and the id of the i^{th} node of $r^{k,l}$ is

$((-i-1) \bmod l) \cdot k + \lceil \frac{i}{l} \rceil$. $(1 \leq i \leq kl)$.

So for instance the processors of $r^{3,5}$ have successive id's:

12 9 6 3 15 11 8 5 2 14 10 7 4 1 13.

Note $r^{k,l}$ consists of k pieces of l nodes. Each piece consists of a decreasing row of l-1 id's, the last id is larger than every other id in the piece. The pieces are - in a certain sense - mutually placed in a decreasing order: the i'th processor in the j'th piece has a larger id than the i'th processor in the (j+1)'th piece.

We now suppose the ring works fully synchronous: all message transmission times are equal and we ignore time needed for calculations in processors. This means, that when a processor sends a message on a timestep t+1, this message can contain the id's of at most 2t+1 processors: the id of the processor itself, the id's of t processors immediately to the left of the processor and the id's of t processors immediately to the right of the processor. This also means that when for two processors p_o, p_1 the strings consisting of this 2t+1 id's are equally ordered, we may assume that p_o sends a message on timestep t+1 if and only if p_1 sends a message on timestep t+1. (We use that the ring works fully synchronous and the algorithm is a comparison algorithm.)

Let $\Phi(k,l)$ be the smallest number t, such that on the ring $r^{k,l}$ all messages that are sent by the full information algorithm A after time t (under the assumption of synchronicity) contain the value of the maximum (and, of course, many other data).

Claim 2.3.4.3.1. $\forall\, l\ \exists\, c_l\ \forall_k\ \Phi(k,l) \le c_l$.

Proof. Suppose not. Let l be given, such that $\forall\, c\ \exists\, k\ \Phi(k,l) \ge c$. Now we first claim that on every ring $r^{k,l}$ at least the first $\lfloor \frac{1}{4}k\cdot l \rfloor$ timesteps messages are sent with the maximum not in it. This we can see by taking k' such that $\Phi(k',l) \ge \lfloor \frac{1}{4}kl \rfloor$. On the ring $r^{k',l}$ on each of the first $\lfloor \frac{1}{4}kl \rfloor$ timesteps there are messages sent that do not contain the value of the maximum (= $k'\cdot l$). Now use that for each part of the ring $r^{k',l}$ with length $\le 2\cdot\lfloor \frac{1}{4}kl \rfloor +1$ that does not contain the maximum id, there exists an equally ordered part of the ring $r^{k,l}$, that also does not contain the maximum id of this ring (=$k\cdot l$). So on $r^{k,l}$ the first $\lfloor \frac{1}{4}kl \rfloor$ timesteps there are messages sent that do not contain the maximum id.

Finally we use, that for each part of $r^{k,l}$ with length $\le 2\cdot\lfloor \frac{1}{4}kl \rfloor+1$ that does not contain the maximum id, there are at least $\frac{1}{2}k + O(1)$

equally ordered parts. So on each of the first $\lfloor\frac{1}{4}\ kl\rfloor$ timesteps there are at least $\frac{1}{2}k + O(1)$ messages sent, so in total at least approximately $\frac{1}{8}\ k^2l$ messages. When we take l as a constant, and let k grow to infinity, we see that the algorithm costs a quadratic number of messages in the worst case. Contradiction. □

As the algorithm is 'message-driven' (a processor can only send a message upon reception of a message, except for the first timestep), each message m, except those sent on the first timestep, has a 'predecessor' message, that is, the message that 'triggered' the processor to send message m. The processor that sent the message on timestep 1, obtained by taking recursively the predecessor message of m is called the originator of m, the successive messages between the originator of m and m we call the chain of messages of m.

Now look at the last message m, that arrives at the processor with the maximum id, and look at its originator and chain of messages. Claim 2.3.4.3.1. shows that the distance between the originator and the processor with the maximum id is at most c_1 on a ring $r^{k,l}$. As the processor p_{max} with the maximum id must know all id's of all processors after timestep n, there are basically the following possibilities:

- p_{max} is the originator of m, the chain of messages of m "goes around the whole ring": each processor sends one message of this chain; the successive messages have either traveled around the whole ring in positive, or in negative direction. (We say "m has traveled around the ring in positive/negative direction"). (See figure 2.3.4.3.a.).

- the originator of m is p_{max} or a processor with distance $\leq c_1$ to p_{max}; the chain of messages goes from the originator to a node, approximately halfway the ring and then returns. So m can inform p_{max} of the id's of $\frac{1}{2}n + c_1$ processors, all on approximately the same half of the ring (seen from p_{max}). Some other messages(s) must inform p_{max} of the other id's. (See figure 2.3.4.3.b.).

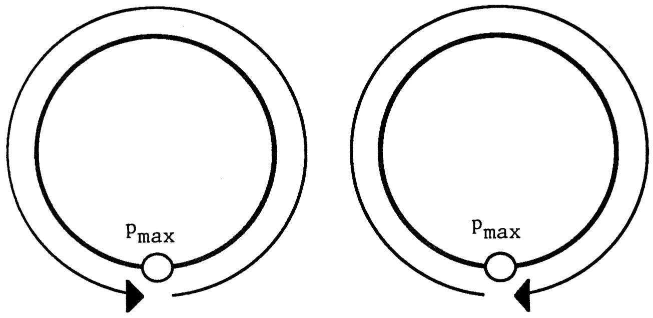


Fig. 2.3.4.3.a.

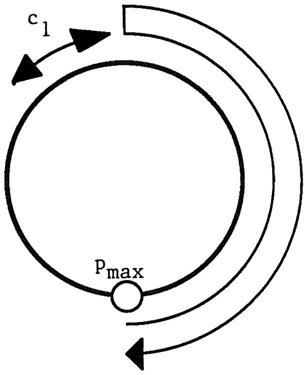


Fig. 2.3.4.3.b.

<u>Claim 2.3.4.3.2</u>. For each l, there are only finitely many k such that the last message m, that is received at the maximum on ring $r^{k,l}$ has traveled around the ring in negative direction.

<u>Proof</u>. Suppose not. For a ring $r^{k,l}$ there are $\frac{1}{2}k + O(1)$ pieces, that are equally ordered to the piece, consisting of the maximum and the $\lfloor\frac{1}{2}k\cdot l\rfloor$ id's of the processors before the maximum. We again use that A is comparison based, hence if on $r^{k,l}$ the last message received at the maximum has traveled around the ring in negative direction, then on each of the first $\lfloor\frac{1}{2}kl\rfloor$ timesteps at least $\frac{1}{2}k + O(1)$ messages are sent, so in total at least approximately $\frac{1}{4}k^2l$. Now we can keep l fixed, and let k grow to infinity: the algorithm uses a quadratic number of messages in the worst-case. Contradiction. □

<u>Claim 2.3.4.3.3</u>. For each l, there is at least one k such that the last message m that is received at the maximum on ring $r^{k,l}$ has traveled around the ring in positive direction.

<u>Proof</u>. Suppose not. Then there are l and k', such that for all $k \geq k'$ the chain of messages of m goes from the originator to a node approximately halfway the ring and then returns. Because the maximum must be informed of the id's of all nodes on the ring, the turning point has a distance of $\frac{1}{2}kl + O(c_l)$. Further notice that if a message chain goes halfway the ring and then back to the maximum on a ring $r^{k,l}$, then a message chain, with a similar length and behavior is sent on $r^{m,l}$, for all $m \geq l$. So when k grows to infinity, the number of messages sent grows at least linear to $\frac{k^2 l}{c_l}$: the algorithm uses a quadratic number of messages in the worst-case. Contradiction. □

Now consider rings $r^{1,l}$, i.e. rings labeled l l−1 l−2... 2 1. The processor labeled l must send a message in positive direction, this message will continue to travel in this direction, for <u>at least</u> the first $\lceil \frac{1}{2}l \rceil$ timesteps. (This follows from claim 2.3.4.3.3, the fact that the algorithm is a comparison algorithm and the fact that during the first $\lceil \frac{1}{2}l \rceil$ timesteps, processors cannot distinguish between the cases that k=1 and k>1.) Because the algorithm is a comparison algorithm, this means that at least $l+l-1 + l-2 + \ldots + \lceil \frac{1}{2}l \rceil$ messages $\approx \frac{3}{8} l^2$ messages are sent on the ring $r^{1,l}$, for every l. Hence the algorithm sends a quadratic number of messages.

We now have shown that every comparison algorithm for problem C that uses time ≤ n must use a quadratic number of messages. Because we have never used in our proof, that the way in which processors decide to send or not to send a certain message must be by an <u>algorithm</u> in the processors, i.e., must be effectively computable by a processor, the results of the previous section show that the obtained result is also valid for arbitrary algorithms. □

The bounds proven in theorem 2.3.4.2. and 2.3.4.3. are of course also valid for problem A: it contains problem C as a subproblem. Theorem 2.3.4.4. shows that these bounds are also valid for problem E.

Theorem 2.3.4.4. Every algorithm for problem E that uses time $\leq$ n must use at least $\frac{1}{2}n(n+1)$ messages in the worst-case for unidirectional rings or $\Omega\ (n^2)$ messages in the worst-case for bidirectional rings.

Proof. First suppose we have a comparison algorithm for problem E that uses time $\leq n$, but uses a smaller number of messages. We will derive a contradiction with theorem 2.3.4.2. and theorem 2.3.4.3. for the unidirectional and the bidirectional case, respectively. We also assume the algorithm is a full-information algorithm. We claim that when a processor knows what id the leader has, then it must know the id's of all the processors. Suppose this is not the case: a processor p decides that the processor with identification number id is the leader (possibly p is self the leader), and the information of p does not contain all id's of all processors. Suppose p knows the id's of k successive processors. We now define a ring r' with 2k nodes. The first k processors have the id's of the successive processors p knows the id of, the second k processors have id's, such that this part of the ring is equally ordered to the first part of k id's. Now the processor p_o with the same id as p, and the processor p_1 with distance k to it can behave similar (the algorithm is a comparison algorithm), so it is possible that p_o and p_1 will both decide on the id of the leader, but then these id's will not be the same. Contradiction.

Because when a processor knows the id's of all the processors it also knows that the maximum id the full-information comparison algorithm can solve problem A in the same time and with the same number of messages. This contradicts theorem 2.3.4.2. or theorem 2.3.4.3. The results of section 2.3.3 show that the obtained bounds are also valid for arbitrary algorithms. □

We assume that the number of bits, needed to express an id is m.

Theorem 2.3.4.5. There exists a unidirectional algorithm for problem B that uses time n, and ≤ 1.356 n log n+O(n) messages in the worst- case, each consisting of O(m+log n) bits.

Proof. Basically we use the algorithm of Dolev, Klawe and Rodeh [DKR82]. To each message we add two fields: one contains the id of the originator, the other the current maximum id known. If a processor id receives a message $\{id_{or}, id_{max}, \text{otherdata}\}$, and $id \neq id_{or}$, then it uses the field 'otherdata' to execute the algorithm of [DKR82]. If this algorithm decides to send a message {newotherdata}, then instead it sends a message $\{id_{or}, id_{max}, \text{newotherdata}\}$. If a processor id receives a message $\{id_{or}, id_{max}, \text{otherdata}\}$ and $id=id_{or}$, then id_{max} is the maximum id of all the processors. It is easy to see that the algorithm works correctly, uses time n, does not use more more messages than the algorithm of [DKR82], and has a message size of O(m+log n) bits. □

Note that the use of the algorithm of [DKR82] is not essential in the proof. For instance, when a better upperbound is found for unidirectional extrema finding, then this upperbound is also valid for problem B and algorithms that use time n.

Theorem 2.3.4.6. There exists a unidirectional algorithm for problem E that uses time n and 1.356 n log n + O(n) messages in the worst-case.

Proof. Use the full-information variant of the algorithm of Dolev, Klawe and Rodeh [DKR82], during the first n timesteps. When a processor receives a message that contains its own id in the first field, then it knows the successive id's of all the processors, hence it can decide what processors also receive messages that contain their own id in the first field and whether it has the largest id of this set of processors. Is this the case, then it declares itself as a leader. In this way in time n exactly one processor will declare itself as a leader; the algorithm does not use more messages than the algorithm of [DKR82]. □

Note that the size of the messages becomes as large as $O(n\cdot m)$ bits. It is an open question whether there exists an algorithm for problem E that uses time n, and $O(n \log n)$ messages of $O(m + \log n)$ bits each in the worst-case.

2.3.5. Lowerbounds for algorithms on rings with known ring size. In this section we consider the extrema-finding problem on rings "that know the ring size", i.e. the number of processors n is initially known to the processors. For this problem a worst-case lowerbound of $\frac{6}{5\log 5}\, n \log n + O(n) \approx 0.51\, n \log n + O(n)$ messages was proved by Pachl, Korach and Rotem [PKR82]. We will improve on this result by looking at comparison algorithms.

To start our analysis, again we replace an algorithm by its full-information variant. Notice that if a processor receives a message with length n−1 then it knows all the id's of the processors, so it also knows the maximum. So now the notion of exhaustiveness is replaced by the following definition:

Definition. Let $E \subseteq D$. E is exhaustive for known ring size n, iff

(i) $\forall\, t,u \in D$: $tu \in E \Rightarrow t \in E$

(ii) $\forall\, s \in D$ with $len(s)=n$: $\exists\, t \in C(s)$: $\exists\, u, v \in D$: $t=uv$, $len(u) = n-1$, $len(v) = 1$, $u \in E$.

Definition. Let $E \subseteq D$. E is comparison-exhaustive for known ring size n, iff E is exhaustive for known ring size n and E is comparison based.

Again we let $N(S,E)$ denote $|\,\{t \in E \mid t$ is a subsequence of an element of $C(S)\}\,|$. We let $K(n)$ ($K^{cb}(n)$) denote the worst-case number of messages sent by the "best" algorithm (comparison algorithm) for rings with known ring size n.

Definition. $K(n) = \min \{ \max_{\substack{s\in D\\ len(s)=n}} N(s,E) \mid E \subseteq D$ is exhaustive for known ring size n}.

$K^{cb}(n) = \min \{ \max_{\substack{s\in D\\ len(s)=n}} N(s,E) \mid E \subseteq D$ is comparison exhaustive for known ring size n}.

Theorem 2.3.5.1.

(i) $K(n)$, $K^{cb}(n)$ are lowerbounds for the worst-case number of messages sent by unidirectional algorithms for known ring size n.

(ii) $K(n) = K^{cb}(n)$.

Proof.

(i) This follows from the definitions. Compare with section 2.3.2.

(ii) Compare with theorem 2.3.3.2 and 2.3.3.4. □

Lemma 2.3.5.2. $K(5) = K^{cb}(5) \geq 12$.

Proof. Let E be comparison exhaustive for known ring size 5. If <1,2> and <2,1> ∈ E then N(<1,2,3,4,5>,E) ≥ 12: <1>, <2>, <3>, <4>, <5>, <1,2>, <2,3>, <3,4>, <4,5>, <5,1> ∈ E (use that E is comparison based), and at least one string with length 3 and one string with length 4, that are substrings of an element of C(<1,2,3,4,5>), must be an element of E. Without loss of generality suppose <1,2> ∈ E, <2,1> ∉ E. If <1,2,3> ∈ E then N(<1,2,3,4,5>) ≥ 13. (Use again that E is comparison based.) So assume <1,2,3> ∉ E. There must be a substring with length 3 of an element of C(<1,2,3,4,5>) an element of E, so <4,5,1> ∈ E. Also <1,5,4> ∈ E. (Use ring <5,4,3,2,1>. <3,2,1> ∉ E, because <3,2,1> ∈ E ⇒<3,2> ∈ E ⇒ <2,1> ∈ E. Etc.)

Now consider the ring labeled <3,5,2,4,1>. <3,5,2> ∈ E, <2,4,1> ∈ E and <5,2,4> ∉ E, <4,1,3> ∉ E, <1,3,5> ∉ E. This means that <3,5,2,4> ∈ E or <2,4,1,3> ∈ E. Because <3,5,2,4> ≡ <2,4,1,3>, both are elements of E. Hence N(<3,5,2,4,1>,E) ≥ 5+3+2+2=12. So $\max_{\substack{s \in d \\ \text{len}(s)=n}} N(5,E) \geq 12$. □

Lemma 2.3.5.3. $\forall\, k,n \in N^{+}$: $K^{cb}(kn) \geq k\, K^{cb}(n) + n\, K^{cb}(k) - kn + k+n-1$.

Proof. Let π^{n} be a permutation of the elements {1,...,n}, and let π^{k} be a permutation of the elements {1,...,k}. Let $\pi^{k} \circ \pi^{n}$ be the permutation of {1,...,kn}, defined by

$(\pi^k \circ \pi^n)(i) = n\cdot\pi^k(((i-1) \bmod k)+1) + \pi^n(\lceil \frac{i}{k} \rceil)$ $(1 \le i \le k-n)$.

In this way the ring labeled by the successive values of $\pi^k \circ \pi^n$ consists of n pieces of k elements. Every piece of k successive elements is order equivalent to an element of $C(\pi^k)$; the relative ordering we obtain by dividing the ring in n pieces of k elements each and then comparing these pieces is given by π^n.

We now claim, that for every (full-information) comparison algorithm A the worst-case number of messages that is sent by A, taken over all rings $\pi^k \circ \pi^n$ is at least $k\cdot K^{cb}(n) + nK^{cb}(k)-kn+k+n-1$.

First note that we obtain a correct (full-information) comparison algorithm for rings with known ring size k, by letting A^k send a message, iff it has length at most k-1 and is sent by algorithm A. Hence A sends at least $nK^{cb}(k)$ messages with length $\le k-1$. It is easy to see that A sends $\ge n$ messages with length k. Now, for each π^k, let A^{n,π^k} be the full information algorithm for rings with size n, that sends a message $s=<s_1,\ldots,s_m>$ $(m \le n)$, iff A sends a message $t=<t_1,\ldots,t_r>$ $mk \le r \le n\cdot k$ with t is a subsequence of an element of $C(\pi^k \circ \pi^n)$ for certain π^n, and $s \equiv <t_k,t_{2k},t_{3k},\ldots,t_{mk}>$. One can check that A^{n,π^k} is a correct algorithm; let the number of messages sent by A^{n,π^k} be α, then the number of messages sent by A with length between k+1 and (n-1)k (inclusive) is at least $k\cdot\alpha-kn \ge kK^{cb}(n) - kn$. Finally, for each i, with $(n-1)k+1 \le i \le nk-1$, A will send at least one message with length i.

The total number of messages is also at least $kK^{cb}(n)+nK^{cb}(k)-nk+k+n+1$. □

A similar, but weaker result was proved by Pachl, Korach and Rotem [PKR82].

From lemma 2.3.5.3. it follows that for any fixed $k \ge 1$, and infinitely many n:

$$K(n) \ge \frac{(K(k)-k+1)n \log n}{k \log k} + O(n).$$

So now we have proved:

Corollary 2.3.5.4. There are infinitely many n, for which the worst-case number of messages, sent by any unidirectional algorithm on a ring with known ring size n, is at least $\frac{8}{5\log 5}$ n log n + O(n) ≈ 0.689 n log n + O(n).

This result improves the lowerbound of $\frac{6}{5\log 5}$ n log n + O(n) ≈ 0.51 n log n + O(n) messages, proved by Pachl, Korach and Rotem [PKR82], and is quite close to the best known lowerbound for the worst-case number of messages in the case that the ring size is not known (i.e. $nH_n \approx$ 0.693 n log n +O(n)).

Formerly no results were known on lowerbounds for the average number of messages for algorithms that can use a known ring size. We will prove a result for comparison algorithms.

Lemma 2.3.5.5. Let E be a comparison exhaustive set for known ring size n. Then:

$$k|n,\ k\neq n \Rightarrow (\forall\ s \in D: \mathrm{len}(s)=k \Rightarrow C(s) \cap E \neq \emptyset\ ).$$

Proof. Let k|n, k ≠ n, s ∈ D, len(s) = k. Because E is comparison based, we may assume without loss of generality, that s ∈ D({1,...,k}), i.e. s can be written as a permutation of the elements {1,...,k}. Now label a ring r as follows:

$$r_i = (s_{i \bmod k}) \cdot \frac{n}{k} + \lceil \frac{i}{k} \rceil . \quad (1 \leq i \leq n).$$

The size of r is n, and for all t ∈ D, with t is a subsequence of an element of C(r) and len(t)=k, there is a u ∈ C(s) with t ≡ S. There must be at least one t ∈ E, with t a subsequence of an element of C(r), and len(t) = k. Now there is a u ∈ C(s), with t ≡ s, t ∈ E, hence u ∈ E, so C(s) ∩ E ≠ ∅ .) □

Theorem 2.3.5.6. Let $n = \prod_{i=1}^{l} p_i$, $p_1, \ldots p_l$ prime numbers. For every unidirectional comparison algorithm A for known ring size n the average number of messages sent is at least:

$$n \cdot \sum_{i=1}^{l} \left(1 - \frac{1}{p_i}\right) - 1$$

Proof. Let E be the comparison exhaustive set for known ring size n, corresponding to A. We can now use a technique similar to a technique used in [PKR82].

We are going to estimate the total number of messages sent in all rings, labeled with permutations of $\{1,\ldots,n\}$. The total number of contiguious label sequences of length k in all these rings is $n(n-1)!=n!$ These can be gathered together in groups of size k, where every group consists of all the cyclic permutations of one sequence. If $k|n$, $k \neq n$, then E intersects each of these groups (lemma 2.3.5.5.) So $k|n$, $k \neq n$ implies that there are at least $\frac{n(n-1)!}{k}$ messages with length k sent.

Furthermore, notice that on any ring, for $k_1 \leq k_2$ the number of messages with length k_1 that are sent is at least the number of messages with length k_2 that are sent. There must also be at least one message with length n-1 sent.

Let M(k) denote the total number of messages sent with length k, over all rings, labeled with a permutation of $\{1,\ldots,n\}$.

$$\sum_{k=1}^{n-1} M(k) \geq \sum_{j=1}^{l} \sum_{k=1+\prod_{i=1}^{j-1} p_i}^{\prod_{i=1}^{j} p_i} M(k)$$

$$\geq \left[\sum_{j=1}^{l} \left(\prod_{i=1}^{j} p_i - \prod_{i=1}^{j-1} p_i \right) \cdot \frac{n!}{\prod_{i=1}^{j} p_i} \right] - n!$$

$$= \left[\sum_{j=1}^{l} \left(1 - \frac{1}{p_j} \right) n! \right] - n!$$

This means that the average number of messages sent by algorithm A is at least $n \cdot \sum_{j=1}^{l} (1 - \frac{1}{p_j}) - 1$. □

Corollary 2.3.5.7. For every unidirectional comparison algorithm A for known ring size $n=2^l$ ($l \in N$) the average number of messages that is sent is at least $\frac{1}{2}n \log n -1$.

Theorem 2.3.5.8. Let A be a unidirectional algorithm for known ring size $n= \pi_{i=1}^{l} p_i$ ($p_1,\ldots,p_l$ prime numbers.) There exists an infinite set of labels $X \subseteq Z$, such that the number of messages sent, averaged over all rings with size n, labeled with elements of X is at least

$$n \cdot \sum_{i=1}^{l} (1 - \frac{1}{p_i}) - 1.$$

If $n=2^l$ ($l \in N$), then this average number is at least $\frac{1}{2}$ n log n-1.

Proof. It is easy to prove a variant of theorem 2.3.3.2. for known ring size. Now let E be the exhaustive set for known ring size n, corresponding to A, apply the theorem and choose $X=A_n$. Then use theorem 2.3.5.6. and corollary 2.3.5.7. □

CHAPTER THREE

DISTRIBUTION OF RECORDS ON A RING OF PROCESSORS

3.1. Introduction. Let n processors be connected in a ring, and assume that each processor can directly communicate with its two immediate neighbors. We usually assume the ring to be oriented. The nodes and links of the network are assumed to work fault- and error free. In this chapter we consider a special version of the load-distribution problem for rings of processors, modeled in the following way (cf. fig. 3.1.1). Assume the network knows rounds. In each round the following sequence of events can/must happen:

- an arbitrary processor receives a packet of information (a "record") from some external source that must be stored somewhere on the ring,
- by some protocol the record is moved to a processor somewhere on the ring (possibly it is not moved at all),
- the record is stored at this processor, and
- (possibly) some administrative actions take place.

There are no a priori constraints on the processors where a single record can be stored. The load-distribution problem asks for protocols that attempt to minimize the maximum number of records stored at a node and/or the number of messages needed to achieve a fair load-distribution on the average. In this chapter several token-based protocols will be proposed and analyzed for the problem, that are of sufficient simplicity to be useful in practice.

For the average case analysis of the protocols (viewed as communication algorithms) we assume that in each round each processor has an equal probability to receive the incoming record from an external source; i.e., for each node v and for each round the probability that v receives a record from an external source in that round is 1/n. We let p denote the total number of records stored at the nodes. Initially p will be o.

The model of the load-distribution problem as presented can be used, for instance, in the following manner: the network is used to

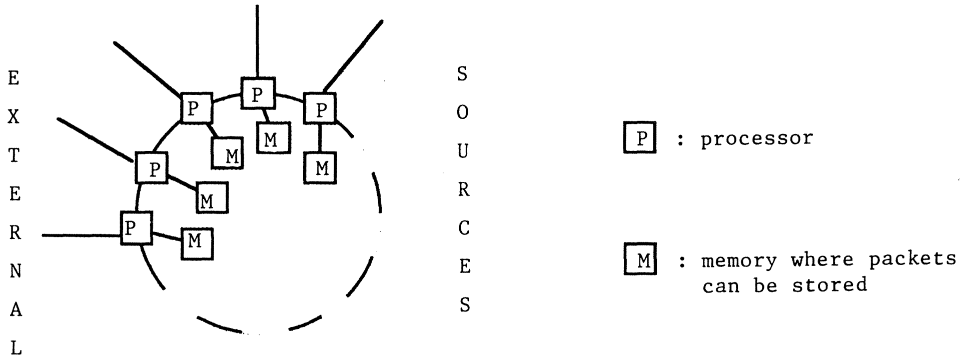


Fig. 3.1.1. A graphical representation of the model

implement a (distributed) database on which insertions of data-records and queries are performed. (In section 3.7. we consider deletions of records as well.) We assume that the number of updates (i.e., insertions) is relatively small over time, and the number of queries is large. Typically queries are sent around the ring and are pipelined, the bottleneck of the pipeline will be the node with the largest number of records stored in memory. Therefore it is desirable to minimize the maximum number of records stored at a node on the ring.

In this chapter we will propose and analyze some token-based protocols for the given model. In section 3.2. we recall some elementary facts of finite Markov chain theory that are used later in the chapter. In sections 3.3., 3.4. and 3.5. we will consider several protocols that use tokens to distribute the records in a (more or less) uniform way over the nodes. In section 3.6. we consider the trivial protocol that stores a record in the very node where it arrives. In section 3.7. we discuss how deletions can be incorporated in the model.

3.2. Elementary facts from the theory of finite Markov chains. In this section we will mention some definitions and results from the theory of finite Markov chains that are needed for the analysis in the later sections. Most of the following definitions and results, and a more detailed introduction in the theory of finite Markov chains, can be found in [Fe68] or [KS60].

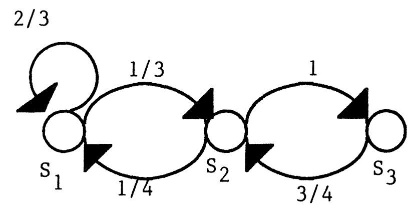
Consider a process S that can be in a finite number of states $s_1,\ldots,s_r$. At each step in time the process can move to another state. Further suppose that the probability that the process moves from state s_i to state s_j only depends on i and j, and on nothing else. We denote this probability by p_{ij}. This type of process is called a finite Markov chain. A finite Markov chain can be represented in two ways:

i) by a (directed) transition graph, with nodes $s_1,\ldots,s_r$ and edges from s_i to s_j labeled p_{ij} for $p_{ij} \neq o$, and

ii) by the <u>matrix of transition probabilities</u>

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & \\ p_{21} & p_{22} & p_{23} & \cdots & \\ p_{31} & p_{32} & p_{33} & \cdots & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ & & & & p_{rr} \end{bmatrix}.$$

For example, consider the following Markov process with 3 states s_1, s_2, and s_3. The probability that the process moves from state s_1 to state s_2 is $\frac{1}{3}$; the probability that the system stays in state s_1 is $\frac{2}{3}$; the probability that the system moves from state s_2 to state s_1 or s_3 is $\frac{1}{4}$ and $\frac{3}{4}$ respectively, and the probability that the system moves from state s_3 to state s_2 is 1 (during one step in time). This process can be represented by the transition graph from fig. 3.2.1.a. and the matrix from fig. 3.2.1.b.



(a)

$$\begin{bmatrix} 2/3 & 1/3 & 0 \\ 1/4 & 0 & 3/4 \\ 0 & 1 & 0 \end{bmatrix}$$

(b)

Fig. 3.2.1.(a) A transition graph and (b) the corresponding matrix of transition probabilities

The probability that the system starts in s_j is denoted by p_j^o. For instance, if it is given that the system starts in state s_i, then p_j^o = o if j ≠ i and $p_i^o = 1$. The probability that the system is in state s_j after t state-transitions from an initial state is denoted by p_j^t. It easily follows that for all j, i≤j≤r and t≥o:

$$p_j^{t+1} = \sum_{i=1}^{r} p_{ij} \cdot p_i^t \qquad (1)$$

$$\sum_{j=1}^{r} p_j^t = 1 \qquad (2)$$

$$o \leq p_j^t \leq = 1 \qquad (3)$$

Let $\vec{p}^{\,t}$ denote the rector $(p_1^t, p_2^t, \ldots, p_r^t)$, and recall that p is the matrix of transition probabilities. From equation (1) it follows that

$$\vec{p}^{\,t+1} = P(\vec{p}^{\,t}), \qquad (4)$$

and hence, $\vec{p}^{\,t} = P^t(\vec{p}^{\,o})$, (5)

for all t ≥ o.

If the transition graph of the Markov chain is strongly connected, i.e. each state can be reached from each other state by a number of transitions with non-zero probability, then the Markov chain is called ergodic. We will further only consider ergodic Markov chains.

Definition. The period of an ergodic Markov chain is the number d=gcd {l| there is a (simple) cycle of length l in the transition graph}.

Suppose the Markov chain has period d. It means that the set of states can be subdivided into d disjunct subjects $S_o, \dots, S_{d-1}$, such that from a state in s_i ($o \le i \le d-1$) only states in $s_{(i+1)\bmod d}$ can be reached by a state-transition with non-zero probability. So the system will be in a state $s_{i_o} \in S_o$, then in a state $s_{i_1} \in S_1$, then in a state $s_{i_2} \in S_2$, etc. If d=1 then the Markov chain is called regular, else it is called cyclic. For all ergodic chains it is valid that the matrix of transition probabilities p has a unique eigenvector $\vec{p} = \langle p_1, \dots, p_r \rangle$ with $\sum_{i=1}^{r} p_i = 1$. For this eigenvector $\vec{p}$ the following will hold:

$$o \le p_i \le 1, \text{ for all } i, 1 \le i \le r \qquad (6)$$

$$P(\vec{p}) = \vec{p} \qquad (7)$$

If the chain is regular (i.e. the period d=1), then p_i denotes the asymptotic probability that the system is in state s_i after t transitions with t growing arbitrarily large.

Theorem 3.2.1. [Fe68,KS60] If d=1, then for all j, $1 \le j \le r$, $\lim_{t \to \alpha} p_j^t = p_j$.

For cyclic chains a slightly weaker result holds. In this case the probabilities p_j denote the asymptotic *average* probability that the system is in state s_j ($1 \le j \le r$), again for the number of state transitions growing arbitrarily large. (The results are also valid for regular chains.)

Theorem 3.2.2. [Fe68,KS60] For all j, 1≤j≤r,

$$\lim_{t \to \alpha} \left(\sum_{i=o}^{t} p_j^i / t \right) = p_j \text{ and}$$

$$\lim_{t \to \alpha} \left(\sum_{i=o}^{d-1} p_j^{(t+1)} / d \right) = p_{j},$$ where d denotes the period of the ergodic Markov chain.

It means that, in order to obtain knowledge over the average asymptotic behavior of an ergodic finite Markov chain, it suffices to find an eigenvector $\vec{p} = <p_1, \ldots, p_r>$ of the matrix of transition probabilities with $\sum_{i=i}^{r} p_i = 1$.

3.3. Token-based protocols for load-distribution with tokens moving in the same direction as records. In the first, elementary protocol we propose for the load-distribution problem, only one token is used. Initially an arbitrary processor is given the token. A record is moved forward on the ring until it arrives at the node holding the token. The record is stored at this node, and the token is passed on to the next node.

Protocol T-1

Each node v has a boolean variable token(v). Initially for exactly one node token(v) is set to true, and for every other node token(v) is set to false.

If v receives a record M either from another node or, at the beginning of a round, from an external source, then v executes:

```
if token(v) then token(v):= false;
                 send a message <token> to the next node;
                 store M
else send M to the next node.
endif
```

```
If v receives a message <token>, then v executes:
    token(v):= true.
```

By protocol T-1 records are distributed uniformly over the nodes; each node either has $\lfloor \frac{p}{n} \rfloor$ or $\lceil \frac{p}{n} \rceil$ records stored. One easily obtains the following bounds.

Theorem 3.3.1. For protocol T-1 the following bounds hold:

i) The maximum number of records stored at a node is $\frac{p-1}{n} + 1$.

ii) The average difference between the maximum number of records stored at a node and p/n is $\frac{1}{2}(1-\frac{1}{n})$.

iii) The maximum number of messages sent in a round is n.

iv) The average number of messages sent in a round is $\frac{1}{2}(n+1)$.

One might want to decrease the number of messages sent by the protocol, by allowing a larger maximum number of records stored at a processor (for a given p). However, while increasing the number of tokens in protocol T-1 will indeed mainly have the effect of increasing the maximum number of records stored at a processor, it will not help much to decrease the (average) number of messages sent by the protocol.

Protocol T-k

Each node v has a counter tokenc(v), which can assume integer values in the range o..k. The value of tokenc(v) represents the number of tokens currently held by v. The values tokenc(v) are initialized such that $\sum_{v=1}^{n}$ tokenc(v) = k, (the sum taken over all nodes v).

If v receives a record M, either from another node or, at the beginning of a round, from an external source, then v executes:

```
if tokenc(v) >o then tokenc(v):= tokenc(v)-1;
                     send a message <token> to the next node;
                     store M
else send M to the next node
endif
```

If a node v receives a message <token>, then it executes:

    tokenc(v):= tokenc(v) +1.

The variables tokenc(v) are maintained so as to denote the number of tokens, present at node v. The total number of tokens will invariantly be k: after each round $\sum_{v=1}^{n}$ tokenc(v) = k. During each round exactly one processor v will execute tokenc(v):= tokenc(v)-1, and exactly one processor v' will execute tokenc(v'):= tokenc(v')+1.

Consider a certain token in some round t. The token will move to the next node in this round if one of the nodes on the path from the node containing the preceding token (this node not included) to and including the node containing the token receives a record from an outside source in this round. The probability of this event is proportional to the length of this path. (See fig. 3.3.1.) This means that tokens will have the tendency to be at small distances from each other. (Consider for example the situation in fig. 3.3.1. If tokens are "close" like tokens 1 and 2 are, at distance ≥ 1, then it is more probable that the first token (i.e. token 1) will move, towards the second one, then that the second token will move away from the first one.) We will analyze this effect of token-clustering for k=2 and even n. A similar analysis is possible for odd n, and k=2. For k≥3 the same effect can be expected.

Suppose k=2, and n is even. We define the ring to be in state s_i ($o \leq i \leq \frac{n}{2}$), if the shortest distance between the nodes with tokenc(v)≥1 is i. In particular, state s_o corresponds to the situation in which there is a node with tokenc(v) = 2 (meaning that the tokens reside in the same node). Denote the probability that the ring is in state s_i after t insertions (i.e., for a total number of packets p=t) by p_i^t. We also assume that for all i, $o \leq i \leq \frac{n}{2}$, p_i^o is given. Naturally $\sum_{i=o}^{n/2} p_i^o = 1$ holds.

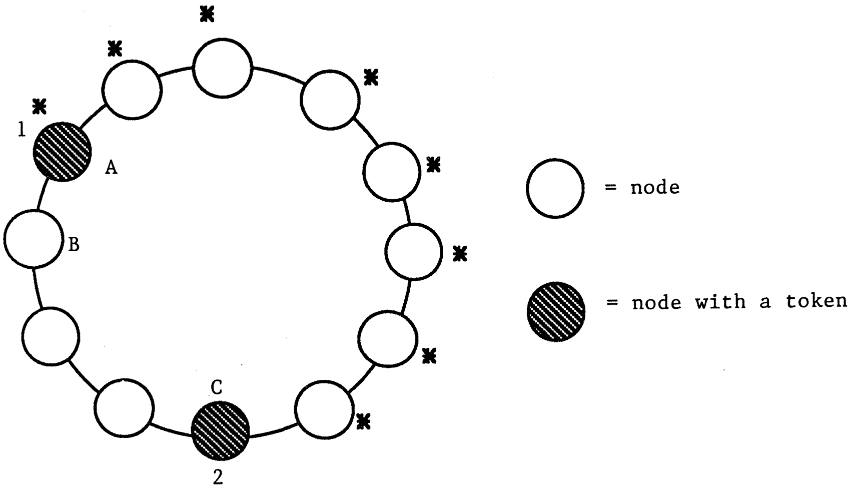


Fig. 3.3.1. If a node with a * receives a record M from an external source, then M is stored in processor A, and token 1 moves up to B in the round, otherwise token 2 moves.

With these definitions the token system on the ring is a finite Markov chain. In fig. 3.3.2. the state-transition graph of the finite Markov chain is given.

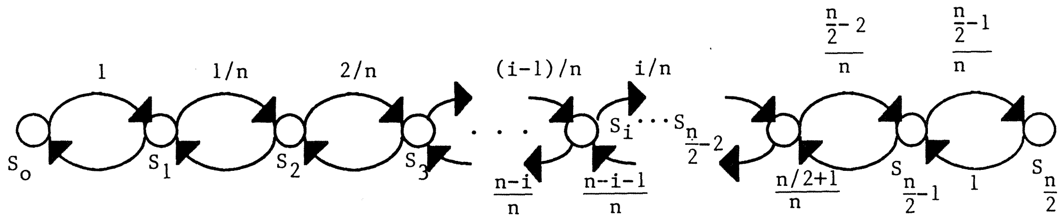


Fig. 3.3.2. The state-transition graph.

The matrix of transition probabilities for this system is the following tridiagonal matrix A:

$$A = \begin{bmatrix}
0 & \frac{n-1}{n} & 0 & 0 \cdots & & & & & & & \cdots & 0 \\
1 & 0 & \frac{n-2}{n} & 0 & & & & & & & & \\
0 & \frac{1}{n} & 0 & \frac{n-3}{n} & & & & & & & & \\
0 & 0 & \frac{2}{n} & 0 & & & & & & & & \\
\cdot & & & & \frac{i-2}{n} & 0 & \frac{n-i}{n} & 0 & 0 \cdots & & & \\
\cdot & & & & 0 & \frac{i-1}{n} & 0 & \frac{n-(i+1)}{n} & 0 & & & \\
\cdot & & & & 0 & 0 & \frac{i}{n} & 0 & \frac{n-(i+2)}{n} & & & \cdot \\
 & & & & 0 & 0 & 0 & \frac{i+1}{n} & 0 & & & \cdot \\
 & & & & & & & & & 0 & \frac{\frac{1}{2}n+1}{n} & 0 \\
\cdot & & & & & & & & & \frac{\frac{1}{2}n-2}{n} & 0 & 1 \\
0 \cdots & & & & & & & & & \cdots 0 & \frac{\frac{1}{2}n-1}{n} & 0
\end{bmatrix}$$

Lemma 3.3.2.

i) $p_o^{t+1} = \frac{n-1}{n} p_1^t$

ii) $p_1^{t+1} = p_o^t + \frac{n-2}{n} p_2^t$

iii) $p_i^{t+1} = \frac{i-1}{n} p_{i-1}^t + \frac{n-(i+1)}{n} p_{i+1}^t \quad (2 \leq i \leq \frac{n}{2}-2)$

iv) $p_{\frac{n}{2}-1}^{t+1} = \frac{\frac{1}{2}n-2}{n} p_{\frac{n}{2}-2}^t + p_{\frac{n}{2}}^t$

v) $p_{\frac{n}{2}}^t = \frac{\frac{1}{2}n-1}{n} p_{\frac{n}{2}-1}^t$.

One easily sees that the Markov chain is ergodic (the graph of fig. 3.3.1. is strongly connected), and that it is cyclic with period 2. We will now look for an eigenvector $\vec{p} = (p_o, \ldots, p_{n/2})$ of A with $\sum_{i=o}^{\frac{n}{2}} p_i = 1$.

We have the following equations for the eigenvector $\vec{p}$:

$$p_o = \frac{n-1}{n} p_1 \tag{1}$$

$$p_1 = p_o + \frac{n-2}{n} p_2 \tag{2}$$

$$p_i = \frac{i-1}{n} p_{i-1} + \frac{n-(i+1)}{n} p_{i+1} \quad (2 \leq i \leq \tfrac{1}{2} n-2) \tag{3}$$

$$p_{\frac{1}{2}n-1} = \frac{\frac{1}{2}n-2}{n} \; p_{\frac{1}{2}n-2} + p_{\frac{1}{2}n} \tag{4}$$

$$p_{\frac{1}{2}n} = \frac{\frac{1}{2}n-1}{n} \; p_{\frac{1}{2}n-1} \tag{5}$$

We will estimate p_o and p_1.

Lemma 3.3.3. For every i with $2 \leq i \leq \frac{n}{2} -1$, $p_i = \frac{i-1}{n-1} p_{i-1}$ and (hence) $p_i = \frac{1}{\binom{n-2}{i-1}} p_1$.

Proof. Use induction on i. For i=2 the lemma follows directly :

$$p_1 = p_o + \frac{n-2}{n}\, p_2 \text{ and } p_o = \frac{n-1}{n} p_1 \Rightarrow \frac{n-2}{n}\, p_2 = \frac{1}{n} p_1 \Rightarrow p_2 = \frac{1}{n-2}\, p_1 .$$

Now suppose the lemma holds for a certain i, $2 \le i \le \frac{n}{2} - 1$. Then $p_i = \frac{i-1}{n}\, p_{i-1} + \frac{n-(i+1)}{n}\, p_{i+1}$ and $p_i = \frac{i-1}{n-i}\, p_{i-1} \Rightarrow \frac{n-(i+1)}{n} = p_i - \frac{i-1}{n} \cdot \frac{n-i}{i-1}\, p_i = \frac{1}{n}\, p_i \Rightarrow p_{i+1} = \frac{i}{n-(i+1)}\, p_i$, and the induction step is complete. The second expression for p_i follows likewise. □

Lemma 3.3.4. For every i,j with $3 \le i \le j \le \frac{n}{2}$, $p_i \ge p_j$.

Proof. The result follows from lemma 3.3.3. and equation (5). □

Lemma 3.3.5.

i) $p_1 = \frac{1}{2} - O(\frac{1}{n^2})$.

ii) $p_o = (\frac{n-1}{n}) . \frac{1}{2} - O(\frac{1}{n^2})$.

Proof.

$$\text{i) } 1 = \sum_{i=o}^{\frac{1}{2}n} p_i = \frac{n-1}{n}\, p_1 + p_1 + \frac{1}{n-2}\, p_1 + \frac{1}{\left[\genfrac{}{}{0pt}{}{n-2}{2}\right]}\, p_1 + \sum_{i=4}^{\frac{1}{2}n} p_i$$

$$\le \left[2 - \frac{1}{n} + \frac{1}{n-2} + \frac{2}{(n-2)(n-3)} \right] p_1 + (\tfrac{1}{2}n-3) \cdot p_4$$

$$= \left[2 - \frac{2}{n(n-2)} + \frac{2}{(n-2)(n-3)} + (\tfrac{1}{2}n-3) \cdot \frac{6}{(n-2)(n-3)(n-4)} \right] p_1$$

$$= \left[2 + O(\frac{1}{n^2}) \right] \cdot p_1, \text{ hence } p_1 = \frac{1}{2+O(\frac{1}{n^2})} = \frac{1}{2} - O(\frac{1}{n^2}).$$

ii) Use (i) and equation (1). □

Lemma 3.3.5. shows that $p_o + p_1$ $\Big[$ = the probability that the two tokens are in the same node (p_o) or in neighboring nodes (p_1)$\Big]$ grows to 1 as n and p/n become arbitrarily large. For instance, if n=20, then one can show (as in lemma 3.3.3.) that $p_1 > 0.4950$ and $p_o > 0.4702$. It means that in the case that n=20, in the long run, on the average, at least 96% of the time the distance between the two tokens is o or 1. This clearly proves the effect of "token-clustering".

<u>Theorem 3.3.6</u>. Let n be even. Assume in protocol T-2 that the tokens start in opposite nodes, i.e. $p^{o}_{\frac{n}{2}} = 1$. The following bounds are valid for protocol T-2.

i) The maximum number of records stored at a node, in worst-case, is $\frac{p}{n} + 1\frac{1}{2} - \frac{2}{n}$.

ii) The average difference between the maximum number of records stored at a node and p/n approaches $1\frac{1}{4}$, as n and p/n tend to infinity.

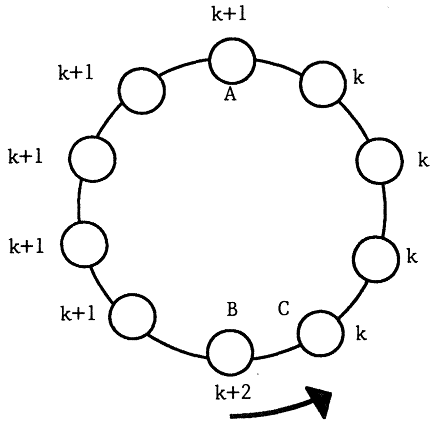
iii) The worst-case number of messages sent in a round is n.

iv) The average number of messages sent in a round approaches $\frac{1}{2}n + O(1)$ as n and p/n tend to infinity.

<u>Proof</u>.

i) The maximum arises when both tokens are in the same node, that is a node, neighboring a node where a token started in round o. Then $\frac{n}{2}$ nodes have k records stored, $\frac{n}{2}-1$ nodes have k+1 records stored and one node (the node where the tokens reside) has k+2 records stored, for certain constant k. (Cf. fig. 3.3.3.). Now $p = \frac{n}{2}\cdot k+(\frac{n}{2}-1)(k+1)+k+2$, and hence the maximum number of records stored at a node is $k+2 = \frac{p}{n} + 1\frac{1}{2} - \frac{2}{n}$. With some straightforward analysis one obtains that no other case gives a larger maximum.

ii) Notice that for our analysis we may assume that the system stays in the set of states $\{s_o, s_1\}$. Suppose the system is in state p_o and the distance between a node where a token started and the node where the tokens reside is i, with $o \le i \le \frac{n}{2}$. With an analysis similar to (i), one obtains that the maximum number of records stored at a node is $\frac{p}{n} + 1 + \frac{i}{n} + O(\frac{1}{n})$. A similar bound can be obtained for the case that the system is in state p_1. As i will be uniformly distributed over the range $<0,1,\ldots,\frac{n}{2}-1>$, the average difference between the maximum number of records stored at a node and $\frac{p}{n}$ will approach $1 + \sum_{i=o}^{\frac{1}{2}n-1} \frac{i}{n} + O(\frac{1}{n}) = 1\frac{1}{4} + O(\frac{1}{n})$.



A: node where token 1 started
B: node where token 2 started
C: node where both tokens reside
Each number denotes the number of records stored at the corresponding node.

Fig. 3.3.3. The maximum number of records stored at a node with protocol T-2.

iii) The worst-case number of messages n is sent in the case that the system is in state s_o, and the record arrives in the node following the node where the two tokens reside. The record must make (n−1) steps until it arrives in a node with a token; one extra message is needed to send the token.

iv) Note that the average number of messages needed in a round with the system in state s_i is $\frac{1}{n}\left(\sum_{j=1}^{i} j + \sum_{j=1}^{n-i} j\right) = \frac{1}{n}\left(\frac{1}{2}(i+1)i+\frac{1}{2}(n-i)(n-i+1)\right) = \frac{1}{2}n+\frac{i^2}{n}-i+\frac{1}{2}$. As n and p/n tend to infinity, the probability that the system is in state p_o approaches $\frac{1}{2}$, the probability that the system is in state p_1 approaches $\frac{1}{2}$, and the probability that the system is in a state p_i with i≥2 approaches o. So the average number of messages will approach $\frac{1}{2}(\frac{1}{2}n+\frac{1}{2})+\frac{1}{2}\left(\frac{1}{2}n+\frac{1}{n}-1+\frac{1}{2}\right)=\frac{1}{2}n+\frac{1}{2n}$. □

The case that n is odd can be analyzed similarly, and yields similar results. Theorem 3.3.6. indicates that in order to decrease the average number of messages sent in a round, other approaches to fair load-distribution must be followed.

3.4. Token-based protocols for load-distribution with tokens moving in the opposite direction as packets. The effect of token clustering that arises in protocol T-k ($k\geq2$) can be avoided in the following, simple manner: let tokens move in the opposite direction as packets. Protocols T′-1 and T′-k are obtained from T-1 and T-k by replacing the statement "send a message <token> to the next node" by "send a message <token> to the preceding node". It is easily seen that for protocol T′-1 exactly the same bounds hold as for protocol T-1 (cf. theorem 3.3.1.).

The main result of this section concerns the asymptotic analysis of the average performance of the protocols T′-k for $k\geq2$, with finite Markov chain theory. The protocols appear to have a fairly ideal behavior. Not only the effect of token-clustering is avoided, but tokens that have a small distance to each other will have the tendency of increasing the distance to each other. The analysis shows that the protocols T′-k indeed decrease the (asymptotic) average number of messages required for load-distribution, while the maximum number of records stored at a node (by a given p) shows only a very small increase.

Suppose the records travel in counter-clockwise direction on the ring. Thus, tokens are sent in clockwise direction according to the instructions of protocol T′-k. Number the tokens consecutively, in counter-clockwise direction (i.e. the direction in which records travel), from 1 to k. Let a_i^t denote the distance of token i to token (i+1) after t rounds, measured along the ring in the same direction, a_k^t denotes the distance of token k to token 1. Note that always $\sum_{i=1}^{k} a_i^t=n$. If, after the t′th round, $a_i^t=a_i$ for $1\leq i\leq k$ and certain a_i ($1\leq i\leq k$) with $\sum_{i=1}^{k} a_i = n$, then the system is said to be in state $S_{a_1 \ldots a_k}$.

The probability that the system is in state $S_{a_1 \ldots a_k}$ after t rounds is denoted by $p^t_{a_1 \ldots a_k}$.

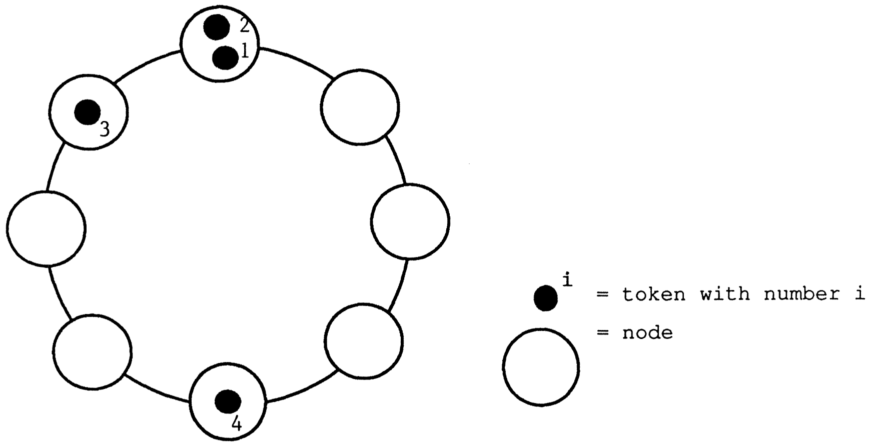


Fig. 3.4.1. A system in state $S_{0,1,2,5}$. (k=4, n=8).

If the system is in state $S_{a_1\ldots a_k}$ with $a_i \geq 1$, then the (i+1)st token can move if one of the nodes on the path from the node, next to the node containing the i'th token, to the node containing the (i+1)st token receives a record M in this round from an external source. The probability of this event is precisely $\frac{a_i}{n}$. As a result of the event $a_{i+1}^{t+1} = a_{i+1}^{t}+1$ and $a_i^{t+1} = a_i^{t}-1$, and the resulting state is $S_{a_1\ldots a_{i-1}a_{i+1}+1\ldots a_k}$. (If i=k, then $S_{a_1\ldots a_i-1a_{i+1}+1\ldots a_k}$ denotes $S_{a_1+1a_2\ldots a_{k-1}a_k-1}$.)

Theorem 3.4.1.

i) For all $a_1,\ldots,a_k \geq o$ with $\sum_{i=1}^{k} a_i = n$ and $t \geq o$,

$$p^{t+1}_{a_1\ldots a_k} = \sum_{\substack{1\leq i\leq k\\ a_{i+1}\geq 1}} p^{t}_{a_1\ldots a_i+1a_{i+1}-1\ldots a_k} \cdot \frac{a_{i+1}}{n}.$$

ii) For all $t \geq o$, $\sum p^{t}_{a_1,\ldots a_k} = 1$, where the summation is taken over all $a_1,\dots,a_k \geq o$ with $\sum_{i=1}^{k} a_i = n$.

The system of tokens is seen to be a finite Markov chain, which is ergodic and has period k. For the asymptotic average behavior of the system we have to solve the following system of equations characterizing an eigenvector of the matrix of transition probabilities as required:

$$-\ p_{a_1\dots a_k} = \sum_{\substack{1\leq i\leq k \\ a_{i+1} (\geq 1}} p_{a_1\dots a_i+1 a_{i+1}-1\dots a_k} \cdot \frac{a_{i+1}}{n}$$

for all $a_1,\dots,a_k \geq o$ with $\sum_{i=1}^{k} a_i = n$ (1)

(*)

$-\ \sum p_{a_1\dots a_k} = 1$, where the summation is taken over all $a_1,\dots,a_k \geq$ o with $\sum_{i=1}^{k} a_i = n$. (2)

From Markov chain theory it follows that this system of equations will have exactly one solution, and the $p_{a_1\dots a_k}$ characterized by the system of equations denote the asymptotic average probability that the system is in state $S_{a_1\dots a_k}$.

Theorem 3.4.2. A solution to the system of equations (*) is:

$$p_{a_1\dots a_k} = \frac{n!}{a_1!\dots a_k!}\,\frac{1}{k^n}, \text{ for all } a_1,\dots,a_k \geq o \text{ with } \sum_{i=1}^{k} a_i = n.$$

Proof. Use the well-known fact that $\sum_{\substack{o\leq a_1\dots a_k \\ \sum_{i=1}^{k} a_i=n}} \frac{n!}{a_1!\dots a_k!} = k^n$.

Now equation (2) follows immediately. Further we have for all $a_1,\dots,a_k \geq o$ with $\sum_{i=1}^{k} a_i=n$

$$\sum_{\substack{1\leq i\leq k \\ a_{i+1}\geq 1}} \frac{n!}{a_1!\dots(a_i+1)!(a_{i+1}-1)!\dots a_k!}\,\frac{a_i+1}{n} =$$

$$\sum_{i=1}^{k} \frac{n!}{a_1!\ldots a_i!\; a_{i+1}!\ldots a_k!} \frac{a_{i+1}}{n} = \frac{n!}{a_1!\ldots a_k!} \cdot \sum_{i=1}^{n} \frac{a_{i+1}}{n} = \frac{n!}{a_1!\ldots a_k!} . \quad \square$$

Theorem 3.4.3. The average number of steps a record must go according to protocol T'-k until it arrives in a node with a token is equal to $\frac{1}{2k}(n-1)$.

Proof. If the system is in state $s_{a_1\ldots a_k}$, then the average number of steps a record must go (in that round) until it arrives in a node with a token is

$$\frac{1}{n} \cdot \sum_{i=1}^{k} \sum_{j=o}^{a_i-1} j = \sum_{i=1}^{k} \frac{a_i(a_i-1)}{2n}.$$

So the (total) average number of steps a record must go is equal to:

$$\sum_{\substack{\left\{a_1\ldots a_k \geq o\right\} \\ \sum_{i=1}^{k} a_i=n}} p_{a_1\ldots a_k} \cdot \sum_{j=1}^{k} \frac{a_j(a_j-1)}{2n} =$$

$$\sum_{\substack{\left\{a_1\ldots a_k \geq o\right\} \\ \sum_{i=1}^{k} a_i=n}} \frac{n!}{a_1!\ldots a_k!} \cdot \frac{1}{k^n} \cdot \sum_{j=1}^{k} \frac{a_j(a_j-1)}{2n} =$$

$$\sum_{j=1}^{k} \sum_{\substack{\left\{a_1\ldots a_k \geq o\right\} \\ \sum_{i=1}^{k} a_i=n}} \frac{n!}{a_1!\ldots a_k!} \cdot \frac{1}{k^n} \frac{a_j(a_j-1)}{2n} =$$

$$k \cdot \sum_{\substack{\left\{a_1\ldots a_k \geq o\right\} \\ \sum_{i=1}^{k} a_i=n}} \frac{n!}{a_1!\ldots a_k!} \cdot \frac{1}{k^n} \cdot \frac{a_1(a_1-1)}{2n} =$$

$$\sum_{\substack{\left\{a_1 \geq 2\right\} \\ a_2 \ldots a_k \geq o \\ \sum_{i=1}^{k} a_i = n}} \frac{(n-2)!}{(a_1-2)!a_2!\ldots a_k!} \cdot \frac{1}{k^{n-1}} \cdot \frac{n-1}{2} =$$

$$\sum_{\substack{\left\{a_1 \ldots a_k \geq o\right\} \\ \sum_{i=1}^{k} a_i = n-2}} \frac{(n-2)!}{a_1!\ldots a_k!} \cdot \frac{1}{k^{n-1}} \cdot \frac{n-1}{2} =$$

$$k^{n-2} \cdot \frac{1}{k^{n-1}} \cdot \frac{n-1}{2} = \frac{1}{2k(n-1)}. \quad \square$$

Consider theorem 3.4.3. for the situation that $k|n$ and the system is in state $S_{\frac{n}{k}\frac{n}{k}\ldots\frac{n}{k}}$, i.e., the tokens are equally spread out over the ring at distance $\frac{n}{k}$. In this "best possible case", the average number of steps a record has to go until it arrives in a node with a token is $\frac{1}{2}(\frac{n}{k}-1)$, which differs less than $\frac{1}{2}$ (!) from the calculated asymptotic average bound for the algorithm T'-k.

Theorem 3.4.4. Assume the system starts in a state $S_{a_1\ldots a_k}$, with $a_i \in \{\lceil\frac{n}{k}\rceil, \lfloor\frac{n}{k}\rfloor\}$ for all i, $1 \leq i \leq k$. The following bounds are valid for protocol T'-k:

i) the worst-case maximum number of records stored at a node is $\frac{p}{n} + \frac{1}{2}k + O(1)$.

ii) the worst-case number of messages sent in a round is n.

iii) the (asymptotic) average number of messages in a round is $\frac{1}{2k}(n-1) + 1$. (n fixed and p tending to infinity).

The assumption in theorem 3.4.4. is only necessary for (i). As we start in a balanced state, it means that also for small round number t, the average number of messages sent in a round will be close to $\frac{1}{2k}(n-1) + 1$. It is currently open to give a good estimate for the (asymptotic) average difference between $\frac{p}{n}$ and the maximum number of records stored at a node.

3.5. Token-based protocols for load-distribution with tokens moving in both directions. In this section we will briefly consider protocols for the load-distribution problem with two types of tokens: tokens that travel in the same direction as the records, and tokens that travel in the opposite direction. We will only analyze the situation with one token of each type. To obtain the best possible bounds, we assume that the tokens start in neighboring nodes, as in fig. 3.5.1.

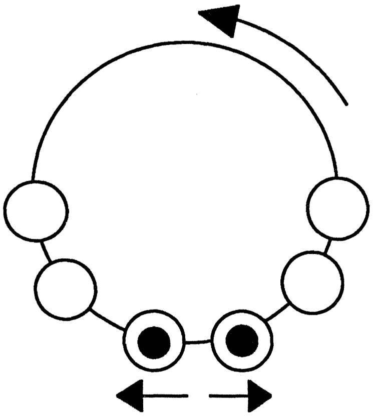


Fig. 3.5.1. Tokens start in neighboring nodes and go in opposite directions.

Protocol TT′.

Each node v has boolean variables left(v), right(v). Initially there is exactly one node v with left(v) = true, all other nodes have left(v) = false, the right neighbor of this node has right(v) = true, all other nodes have right(v) = false.

If v receives a record M, either from another node or, at the beginning of a round, from an external source, then v executes:

```
if left(v) then      left(v):=false;
                     send <left> to left neighbor;
                     store M
elif right(v) then   right(v):=false;
                     send <right> to right neighbor;
                     store M
else send M to right neighbor
endif.
```

If v receives a message <left>, then it executes:

```
    left(v):=true.
```

If v receives a message <right>, then it executes:

```
    right(v):=true.
```

Lemma 3.5.1. After the t'th round the distance of the "left"-token to the "right"-token is (t+1) mod n (measured in the direction of the packets).

Proof. Use induction on t. For the distance between the tokens it is unimportant whether the "left"-token moves or the "right"-token moves. □

Theorem 3.5.2. The following bounds are valid for protocol TT':

i) The worst-case maximum number of records stored at a node is $\frac{p}{n} + (1-\frac{1}{n})$.

ii) The average difference between this maximum and $\frac{p}{n}$ is $\frac{1}{2}(1-\frac{1}{n})$.

iii) The worst-case number of messages sent in a round is n.

iv) The average number of messages sent in a round is $\frac{1}{3}n + \frac{2}{3n} + \frac{1}{2}$, in the long run.

Proof.

i),ii) Each node has either $\lceil\frac{p}{n}\rceil$ or $\lfloor\frac{p}{n}\rfloor$ packets stored.

iii) Trivial.

iv) If the distance of the left token to the right token is i, then on the average a record must go $\frac{i(i-1)}{2n} + \frac{(n-i)(n-i-1)}{2n}$ steps, in that round, until it arrives in a node with a token. If i=o then the average distance is $\frac{1}{2}(n-1)$. From lemma 3.5.1. we have that each of the distances between the tokens o,1,...,n-1 appears equally often (in the long run). So the total average number of steps needed until a record arrives in a node with a token is

$$\frac{1}{n}\sum_{i=o}^{n-1}\left[\frac{i(i-1)}{2n} + \frac{n-i}{2n}(n-i-1)\right] = \frac{1}{n}\left(\sum_{i=o}^{n}\left[\frac{i(i-1)}{n}\right] - \frac{1}{2}(n-1)\right)$$

$$= \frac{n(n+1)(2n+1)}{6n^2} - 1 + \frac{1}{2n} = \frac{1}{3}n + \frac{2}{3n} - \frac{1}{2}.$$

One extra message is needed to move the token. □

Note that each of the bounds for protocol TT' are equal to or better than the corresponding bounds for protocol T-1 or T'-1; the average number of messages is smaller by a factor of about $1\frac{1}{2}$.

3.6. <u>Load-distribution by "direct placing"</u>. In this section we will examine the trivial protocol that just stores a record in the node that receives a record from an external source.

This simple protocol uses no messages; the worst-case number of records stored at a node is p (if packets always arrive in the same node). Thus, the main problem in analyzing this protocol is the expected maximum number of packets stored at a node (for a given p). We can reformulate the problem as follows:

> Suppose we have n urns and p balls. For every ball an urn is chosen at random (with for each urn a probability 1/n that it is chosen). What is the expected number of balls that is in the urn with the largest number of balls?

We denote this expected number by E_n^p. We know of results on related problems (see e.g. [DB62], [Fe68] or [JK77]), but we do not know of results that give estimates or even exact expressions for E_n^p. For even p one can derive an exact expression for E_2^p (i.e., the case of two urns).

Theorem 3.6.1. For p even, $E_2^p = \frac{p}{2} + \binom{p}{p/2} \cdot \frac{p}{2^{p+1}}$.

Proof. The probability that the first urn contains i balls and the second urn contains p-i balls is $\binom{p}{i}/2^p$. So

$$E_2^p = \sum_{i=o}^{p} \left(\frac{\binom{p}{i}}{2^p} \max(i,p-i) \right) = 2 \cdot \sum_{i=p/2+1}^{p} \frac{\binom{p}{i}\ i}{2^p} + \frac{\binom{p}{p/2} \cdot p/2}{2^p}$$

$$=2 \cdot \sum_{i=p/2+1}^{p} \binom{p-1}{i-1} \frac{p}{2^p} + \frac{p}{2^{p+1}} \binom{p}{p/2}$$

$$= \sum_{i=o}^{p-1} \binom{p-1}{i} \frac{p}{2^p} + \frac{p}{2^{p+1}} \binom{p}{p/2} = p/2 + \binom{p}{p/2} \cdot \frac{p}{2^{p+1}}. \quad \square$$

The following results enable us to obtain estimates for E_n^p for all p and n.

Theorem 3.6.2. Let k|n. Then $E_k^p \le \frac{n}{k} E_n^p$.

Proof. Simulate the experiment with k urns in the following way: First carry out the experiment with n urns (and p balls). Then we add together the balls of the first $\frac{n}{k}$ urns, the balls of the next $\frac{n}{k}$ urns, etc. So urn i in the final experiment contains the balls of urns $\frac{n}{k}(i-1)+1$ to $\frac{n}{k} \cdot i$ of the first experiment. Now each ball has a probability $\frac{1}{k}$ to be in the i'th urn, for all i, $1 \le i \le k$; and the choices of urn for the balls are independent of each other. So this is a correct way to simulate the 'E_k^p-experiment'.

If the maximum number of balls in an urn in the experiment with n urns is m, then the maximum number of balls in an urn in the experiment with k urns is at most $\frac{n}{k} \cdot m$. So $E_k^p \le \frac{n}{k} E_n^p$. □

Theorem 3.6.3. Let n be even. Then $E_2^p \geq E_n^p + \frac{p-E_n^p}{n-1}(\frac{1}{2}n-1)$.

Proof. We simulate the experiment with 2 urns as in the proof of theorem 3.6.2. First carry out the experiment with n urns and p balls. Then choose $\frac{n}{2}$ urns at random, and add the balls of these urns together. Also add the balls of the remaining urns together. Again each ball has probabilities $\frac{1}{2}$ to be in the first, or in the second set, respectively, and the choices for the individual balls do not depend on each other. So this is a correct simulation of the experiment with 2 urns and p balls.

Let the balls be numbered 1...p. There is a unique urn X (after the first experiment), with the following properties:

- no urn contains more balls than X does
- no urn that contains the same number of balls as X, contains a ball with a higher number than the highest numbered ball in urn X.

(We need the last constraint to have a unique urn X which contains the maximum number of balls.) Let X contain m balls. The number of balls in the set where X belongs to, averaged over all possible choices of $\frac{n}{2}$ urns is $m + \frac{p-m}{n-1}(\frac{1}{2}n-1)$. (Note that each of the $\frac{1}{2}n-1$ urns $\neq$ X in the set contains on the average $\frac{p-m}{n-1}$ balls). Hence $E_2^p \geq E_n^p + \frac{p-E_2^p}{n-1}(\frac{1}{2}n-1)$. □

Lemma 3.6.4. $E_n^p + \frac{1}{n} \leq E_n^{p+1} \leq E_n^p + 1$.

Proof. First do the E_n^p-experiment, and then add the (p+1)'st ball. The probability that this ball is in an urn with the maximum number of balls (so the ball increases the maximum by 1) is at least $\frac{1}{n}$. The maximum will never be increased by more than 1. □

Definition. For all p, n ≥ 1, let $X_n^p = E_n^p - \frac{p}{n}$.

Theorem 3.6.5. Let n ≥ 2. Then

$$\frac{n-1}{n} X_{n+1}^p \leq X_n^p \leq \frac{n+1}{n} X_{n-1}^p$$

Proof. The experiment with n urns and p balls is simulated as follows: do the "E_{n+1}^p-experiment", and then take urn n+1 and choose for each of its balls randomly one of the first n urns (each urn with probability 1/n). The expected total number of balls in an urn now again is E_n^p.

Number the balls 1...p. Again there is a unique urn X with the following properties:

- no urn contains more balls than X does,
- no urn that contains the same number of balls as X, contains a ball with a higher number than the highest numbered ball in urn X,

with regard to the outcome of the first experiment (with n+1 urns).

The probability that X is one of the first n urns is $\frac{n}{n+1}$. Suppose it is one of the first n urns. The expected number of balls in X is E_{n+1}^p. The expected number of balls in urn (n+1) now is $\frac{p-E_{n+1}^p}{n}$. So, the expected number of extra balls, added to urn X in the last stage of the experiment is $\frac{p-E_{n+1}^p}{n^2}$. This means that the expected number of balls in urn X after the last stage of the experiment is $E_{n+1}^p + \frac{p-E_{n+1}^p}{n^2}$.

If urn X is urn (n+1), then note that the maximum number of balls in an urn after the last stage of the experiment is at least $\frac{p}{n}$.

So $E_n^p \ge \frac{1}{n+1}\,\frac{p}{n} + \frac{n}{n+1}\left(E_{n+1}^p + \frac{p-E_{n+1}^p}{n^2}\right)$

$$= \frac{1}{n+1}\,\frac{p}{n} + \frac{n}{n+1}\left(\frac{p}{n+1} + X_{n+1}^p + \frac{p-\frac{p}{n+1} - X_{n+1}^p}{n^2}\right)$$

$= \frac{p}{n} + \frac{n-1}{n}.X_{n+1}^p$, and hence $X_n^p \ge \frac{n-1}{n}\,X_{n+1}^p$.

Similarly $X_n^p \le \frac{n+1}{n}\,X_{n+1}^p$. □

Theorem 3.6.6. Let p be even.

i) If n is even, then

$$\frac{p}{n} + \frac{1}{n}\,\frac{p}{2^p}\binom{p}{\frac{p}{2}} \le E_n^p \le \frac{p}{n} + \frac{n-1}{n}\,\frac{p}{2^p}\binom{p}{\frac{p}{2}}$$

ii) If n is odd, then

$$\frac{p}{n} + \frac{1}{n+1}\,\frac{p}{2^p}\binom{p}{\frac{p}{2}} \le E_n^p \le \frac{p}{n} + \left(1-\frac{1}{n^2}\right)\frac{p}{2^p}\binom{p}{\frac{p}{2}} .$$

Proof.

i) From theorem 3.6.3. one derives (by elementary formula manipulation) that $x_n^p \le 2.\frac{n-1}{n}.x_2^p$. The desired inequality now follows from this fact and theorems 3.6.1 and 3.6.2.

ii) Use the previous result and theorem 3.6.5. □

For odd p, an estimate of E_n^p follows directly from lemma 3.6.4. Also, note that by using Stirling's approximation formula:

$$e^{-\frac{1}{3}p}\frac{\sqrt{2}}{\sqrt{\Pi}}\sqrt{p} \le \frac{p}{2^p}\binom{p}{p/2} \le \frac{\sqrt{2}}{\sqrt{\Pi}}\sqrt{p}.$$

Theorem 3.6.6. shows for the trivial protocol that the maximum number of records stored at a node will be $\frac{p}{n} + O(\sqrt{p})$, on the average.

3.7. Load-distribution under insertions and deletions. An interesting question is how deletions can be handled. To model this situation we assume that at the beginning of a round two different types of record can arrive: records that have to be inserted (stored at a node), and records that have to be deleted (removed from the node where it was stored earlier). Clearly, for a deletion, one always needs to find the node where the record is stored first: this costs n-1 messages in the worst-case and $\frac{1}{2}n$ messages in the average case. In our analysis we will always assume that one never tries to delete records that are not stored in some node.

As before we let p denote the total number of records that are presently stored on the ring, and we let q denote the number of records that were inserted, but deleted later (and thus are no longer stored). The total number of insertions that has taken place thus equals p+q.

We will consider two approaches to deletions. First we consider the trivial protocol that handles a deletion by removing a record from the node where it is stored, and does nothing further. Next we consider a protocol that also moves records from one node to another node, in order to maintain a (more or less) uniform distribution of the records over the nodes. For the average case analysis we assume that all records are equally likely to be deleted.

First assume the trivial deletion protocol is used in combination with the trivial insertion protocol of section 3.6. It is obvious that records that are inserted and later deleted do not influence the number of records stored at any node after the deletion of the record. So the maximum number of records stored in a node will be E_n^p, which can be estimated as in section 3.6.

Next we consider the case where the trivial deletion protocol is used in combination with one of the protocols of sections 3.3., 3.4. and 3.5. (protocols T-1, T-k, T'-1, T'-k or TT'). For this case we observe the following. Consider the experiment of section 3.6., and let F_n^p denote the expected minimum number of balls in an urn (when n urns and p balls are used in the experiment). Let f(p,n) denote the average difference between p/n and the maximum number of packets at a node with the insertion algorithm that is used, if there were only insertions. We fix a moment t in time, and again let p denote the number of stored records and q the number of records that were once inserted, but deleted later.

Lemma 3.7.1. The expected maximum number of records stored at a node is at most

$$\frac{p+q}{n} + f(p+q,n) - F_n^q .$$

Proof. The expected maximum number of records that are inserted at a node (so are stored presently or were stored previously and were deleted thereafter) equals $\frac{p+q}{n}$ + f(p+q,n). If for each node the probability that a record is deleted from that node is equal, then the expected minimum number of records deleted from that node is equal, then the expected minimum number of records deleted from a node is F_n^q. However, this probability is proportional to the number of records stored at a node, so nodes with a larger number of stored records will have a larger expected number of deleted records. Note that this will result in an expected maximum number of records which will be (slightly) smaller than $\frac{p+q}{n}$ + f(p+q,n) − F_n^q. □

F_n^q can be analyzed similar to E_n^q. We write $F_n^q = \frac{p}{n} - Y_n^p$.

Theorem 3.7.2.

i) Let p be even. $F_2^p = \frac{p}{2} - \frac{p}{2^{p+1}} \binom{p}{p/2}$.

ii) Let k|n. $F_k^p \geq \frac{n}{k} F_n^p$.

iii) Let n be even. $F_2^p \leq F_n^p + \frac{p-E_n^p}{n-1} (\frac{1}{2}n-1)$.

iv) Let $n \geq 2$. $\frac{n-1}{n} Y_{n+1}^p \geq Y_n^p \geq \frac{n+1}{n} Y_{n-1}^p$.

v) $F_n^p + \frac{1}{n} \leq F_n^{p+1} \leq F_n^p + 1$.

Proof. i) Use $F_2^p + E_2^p = p$.
ii), iii), iv), v) Similar to the analysis in section 3.6. □

Theorem 3.7.3. Let p be even.

i) If n is even, then $\frac{p}{n} - \frac{n-1}{n} \frac{p}{2^p} \binom{p}{p/2} \leq F_n^p \leq \frac{p}{n} - \frac{1}{n} \frac{p}{2^p} \binom{p}{p/2}$

ii) If n is odd, then $\frac{p}{n} - (1 - \frac{1}{n^2}) \frac{p}{2^p} \binom{p}{p/2} \leq F_n^p \leq \frac{p}{n} - \frac{1}{n+1} \frac{p}{2^p} \binom{p}{p/2}$.

For odd p, theorem 3.7.2.(v) shows how to obtain estimates for F_n^p with theorem 3.7.3. Again we note that $e^{-\frac{1}{3}p} \frac{\sqrt{2}}{\sqrt{\Pi}} \sqrt{p} \leq \frac{p}{2^p} \binom{p}{p/2} \leq \frac{\sqrt{2}}{\sqrt{\Pi}} \sqrt{p}$. Now we have the following result.

Theorem 3.7.4. The expected maximum number of records stored at a node is at most $\frac{p}{n} + f(p+q,n) + \frac{\sqrt{2}}{\sqrt{\Pi}} \cdot \sqrt{q}$.

Proof. Use lemma 3.7.1.

$\frac{p+q}{n} + f(p+q,n) - F_n^q \leq \frac{p+q}{n} + f(p+q,n) - \left[\frac{q}{n} - \frac{q}{2^q} \binom{q}{q/2}\right] \leq \frac{p}{n} + f(p+q,n) + \frac{\sqrt{2}}{\sqrt{\Pi}} \sqrt{q}$. □

Finally we consider a load-distribution protocol that tries to maintain a balanced distribution of the records over the nodes after each deletion by moving a record from some node to the node where the record was deleted (in general).

The protocol is used in combination with a protocol T′-k, for some k≥1. The tokens used for insertions in algorithm T′-k are also used for deletions. For a deletion of a record M the algorithm proceeds as follows. First the node v is found where the record M is stored, and must be deleted. Then we find the first node preceding v that contains a token. This token is moved to the next node (i.e., in the direction in which records move under insertions). From this (next) node, a record M′ is deleted, M′ is moved to the node v where M was stored and M′ is inserted at v.

Protocol DT-k.

Each node v has a variable tokenc(v), as in protocol T′-k, and a boolean variable gap(v), which is initially false for all nodes v. For insertions protocol T′-k is used.

If v receives a message <delete, M>, either from another node or, at the beginning of a round, from an external source, then v executes:

```
if M is stored at v then delete M,
                         gap(v):=true:
                         send <find token> to preceding node
else send <delete, M> to next node
endif.
```

If v receives a message <find token>, then v executes:

```
if tokenc(v) > o then tokenc(v):= tokenc(v)-1;
                      send <collect> to next node
else send <find token> to preceding node
endif.
```

If v receives a message <collect>, then v executes:

```
    tokenc(v):=tokenc(v) +1;
    if gap(v) then gap(v):=false
    else delete a packet M' from v
           send <fillgap, M'> to next node
    endif.
```

If v receives a message <fillgap, M'>, then v executes:

```
    if gap(v) then store M' at v;
                   gap(v):=false
    else send <fillgap, M'> to next node
    endif.
```

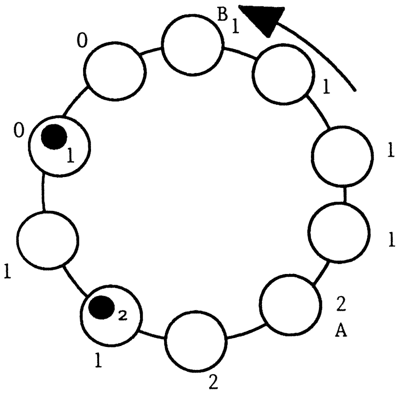
The following properties are invariants for the protocol after each completion of a round:
- gap(v)=false for all nodes v,
- the total number of tokens is k (Σ tokenc(v)=k, where the sum is taken over all nodes v.)

However, to prove the protocol correct we need to show that always a packet M' exists when a collect message is generated. To show this, follow individual tokens as insertions and deletions take place. In each round, one of the tokens moves to a neighboring node. For each node v, let $\Psi(v)$ be the number of tokens for which v lies on the path, starting with the node after the token, upto and including the node where the token initially started, in the direction in which insertion records travel. An example, with 2 tokens is given in fig. 3.7.1.

Lemma 3.7.5. The following property is invariant under algorithm DT-k after each completion of a round:
- there is a $j \geq -k$ such that each node v has exactly $\Psi(v)+j$ records stored.

Proof. First we note the property is initially true: note that $\Psi(v)=k$ for all nodes v.

If in a round t a record is inserted in a node v, then there are two cases:



A: node where token 1 started initially

B: node where token 2 started initially

i: token with number i

Fig. 3.7.1. An illustration of the function Ψ(v). With each node v the number Ψ(v) is shown. The arrow shows the direction in which insertion records move.

- v is not the node where the (moving) token initially started. Then the path, starting with the node after the token to the node where the token initially started, is extended by the node v, hence Ψ(v) increases by 1, and for all other nodes w Ψ(w) does not change. The invariant is again valid after the completion of the round.
- v is the node where the moving token initially started. Then the path contains after the round only the node v, where formerly each node on the ring was on the path (from the node after the token to the node where the token initially started.) So Ψ(w) drops by 1 for all nodes w ≠ v, but Ψ(v) does not change. By increasing j by 1, one sees that the invariant is again valid after completion at the round.

A similar analysis can be given for deletions. □

It follows that if a node v does not contain a token, then it has at least the same number of records stored as the next node. This shows that if a node v receives a <collect> message and not(gap(v)) holds, then there is at least one record stored at v (so there exists a record M′ that can be deleted from v.) Without much difficulty one obtains the following result.

Theorem 3.7.6. Assume the system starts in a state $S_{a_1\ldots a_k}$ with for all i, $1\le i\le k$, $a_i \in \{\lceil \frac{n}{k}\rceil, \lfloor \frac{n}{k}\rfloor\}$, i.e. the distance between each pair of 'neighboring' tokens is $\lceil \frac{n}{k}\rceil$ or $\lfloor \frac{n}{k}\rfloor$. The following bounds are valid for protocol DT-k:

i) The worst-case maximum number of records stored at a node is $\frac{p}{n} + \frac{1}{2}k + O(1)$.

ii) The worst-case number of messages sent in a round with an insertion is n.

iii) the worst-case number of messages sent in a round with a deletion is 3n-1.

The average number of messages needed in a round of protocol DT-k can be estimated in the following way. Suppose that in a certain round t a record is deleted. The probability that a certain token i moves in that round (to the next node) is proportional to the total number of records stored at the nodes on the path from the node with token i to the node, immediately before the node with token i+1. (So if this total number of records is p', then the probability is p'/p.) The numbers $\Psi(v)$ range between o and k, so if p/n becomes large, the probability becomes approximately proportional to the length of the path between the tokens. If we assume that this probability is exactly the length of this path divided by n, then the resulting model can be analyzed similar as in section 3.4. For instance, one can easily derive, that - given that in round t+1 a deletion occurs - for all $a_1\ldots a_k \ge o$

$$p^{t+1}_{a_1\ldots a_k} = \sum_{\substack{1\le i\le k \\ a_i\ge 1}} p^{t}_{a_1\ldots a_i-1a_{i+1}+1\ldots a_k} \cdot \frac{a_{i+1}+1}{n}.$$

As in section 3.4. one can derive that the average probabilities that the system is in state $S_{a_1\ldots a_k}$ approach $\frac{n!}{a_1!\ldots a_k!}\cdot\frac{1}{k^n}$, if the number of insertions (and deletions) becomes arbitrarily large. If follows that an insertion costs approximately n/k +O(1) messages on the average, and a deletion costs $\frac{1}{2}n + \frac{2n}{k} + O(1)$ messages on the average.

CHAPTER FOUR

DEADLOCK-FREE PACKET SWITCHING NETWORKS WITH VARIABLE PACKET SIZE

4.1. Introduction. In packet switching networks the occurrence of deadlock is very undesirable. Therefore one needs controllers (algorithms that control the flow of packets through the network), that prevent the occurrence of deadlock. In this chapter we present a class of controllers that prevent deadlock, use only local information known to the processors and allow packets to have a variable size. Some controllers in this class are proven to be optimal with respect to other controllers using the same local information.

Consider a packet switching network, represented by a (possibly directed) graph G=(V,E). V represents the set of processors, E represents the set of links between processors. Some processors may want to send messages (packets) to other processors in the network. Each packet has to follow a path in G from its source node to its destination node, in accordance to some routing strategy. We make but one assumption on the routing strategy that is followed by a network: there must be an integer k, known to all processors, such that each packet needs to traverse at most k links to arrive at its destination. Note that k is not necessarily at least the diameter of G: it is possible that we do not want to send messages between nodes that have a large distance between them.

During the transportation of a packet from source to destination, the packet has to be stored (queued) temporarily in the memory of the intermediate nodes on the path of this packet. For this purpose each processor has an (integer) number of buffers in which packets can be stored. We assume each node has b buffers, where b is a fixed constant. Each packet p has an (integer) size, denoted by s(p), which is the number of buffers the packet needs when stored at a node. The maximum size of a packet is assumed to be q, so for each packet p one has $1 \le s(p) \le q$. The sum of the sizes of the packets stored at a certain node on a certain moment can never exceed b. We also assume it is possible to store a packet p at node v, if s(p) plus the sum of the sizes of the packets currently stored at v does not exceed b. (Later we will forbid the acceptance of packets p at nodes v in some cases, even when it is theoretically possible to store p at v.) Suppose the buffers of a node v are numbered 1,...,b. In some cases it is necessary to store a packet in buffers, that are not consecutively numbered, or to reallocate one or more packets internally in the memory of the processors. For an example see fig. 4.1.1. Let b=6, $q \geq 2$. At a node v packets with sizes 2, 1 and 2 (in this order) are stored (fig. 4.1.1.a.). The packet with size 1 leaves v, the buffer where it was stored is "free" again (fig. 4.1.1.b.). To store another packet with size 2, we can either reallocate the second packet with size 2 (fig. 4.1.1.c.) or we have to store the packet at non-consecutive buffers (fig. 4.1.1.d.). We will further ignore this problem.

We consider three types of "moves", made in the network:

i) Generation of a packet: a node v generates a packet p and places it in s(p) of its empty buffers.

ii) Passing of a packet: a packet p is passed from a node v to a node w ((v,w) $\in$ E) in accordance to the followed routing strategy; the buffers in v where the packet was stored become empty and the packet is placed in s(p) empty buffers of w.

iii) Consumption of a packet: a packet p that is stored at the buffers of its destination node v is removed from the buffers of v; the s(p) buffers where p was stored become empty.

For our analysis we assume that moves of the network take one unit of time each: each moment t is either before a specific move or after this move, never "during" the move. This is no real limitation of the model, but facilitates the analysis.

A (distributed) algorithm that permits some moves of the network and forbids others is called a controller. An important and very desirable property of controllers is that they prevent deadlock. A deadlock occurs if there is a situation when there are packets which will never arrive at their destination, no matter what sequence of moves is

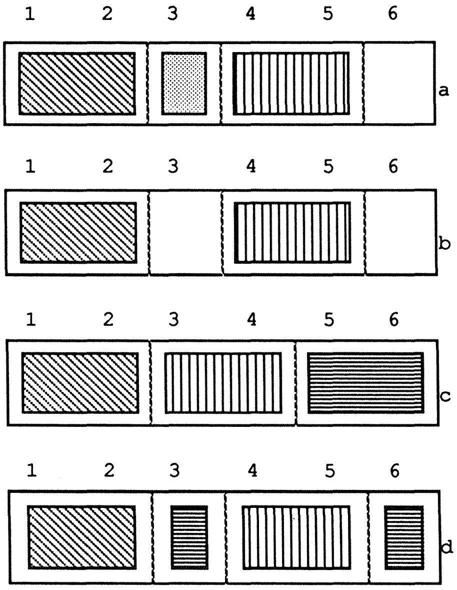


fig. 4.1.1.

performed by the network. A (classic) example of a network with deadlock is given in fig. 4.1.2. Assume the controller permits all moves that do not let the sum of the sizes of the packets stored at v exceed b. If in v_1 b packets are created with size 1 and destination v_2, in v_2 b packets are created with size 1 and destination v_3 and in v_3 b packets are created with size 1 and destination v_1, then the network cannot make another move, so there is a deadlock.

<u>Definition</u>. (Toueg and Ullman [TU81]). A controller is deadlock-free for a given network, if it does not permit the network to enter a state in which one or more packets can never make a move permitted by that controller, as long as no additional packets are generated.

Toueg and Ullman [TU81] noted that the condition that no additional packets are generated is essential to exclude certain strange

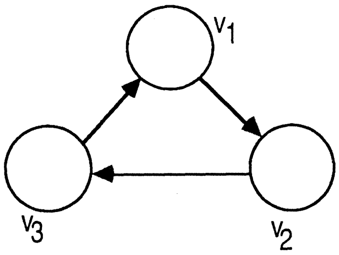


fig. 4.1.2. A network with a controller that does not prevent deadlock.

controllers, where a packets can only move if some other packet is generated, and that packet can only move if a third packet is generated, and so on.

We will only consider local controllers: controllers that use only information about the local state of a node and the state of a packet, but do not use global information. Global controllers have an important drawback: if the size of the network becomes large it can cost much additional work and messages to gain global knowledge over the state of the network.

Several deadlock-free controllers, that use only local information are known. (See e.g. [Ge81], [MS80], [Sh84] and [TU81].) Each of these controllers assumes that packets have a unit size, i.e. $q=1$. Toueg and Ullman [TU81] proposed 4 controllers, that use only the state of a packet p, and the state of the packets stored at node v, to decide whether v can accept p or not, called: forward-count, backward-count, forward-state and backward-state, and proved each to be deadlock-free. We generalize these results in three ways:

- we allow packets to have a variable size,
- we show that "forward-count" and "backward-count" are special cases of the more general notion of "count-down", and thus obtain a uniform theory of deadlock-free controllers of this type, and

- we show that every controller that is "in between" a count-controller and the corresponding state-controller is also deadlock-free.

(For more precise definitions of the notions used here, see section 4.2.) This chapter is organized as follows. In section 4.2 we give some basic definitions. In section 4.3 we present a large class of local deadlock-free controllers. In section 4.4 we show that some of the controllers in this class are optimal in the class of all deadlock-free controllers using the same local information. Some final comments are made in section 4.5.

4.2. Definitions.

Definition. A count-down function Φ is a partial function that maps a triple, consisting of a packet p, a node v and time t to an integer, such that:

i) If p is stored at v on moment t, or v is the next node on the path of p on moment t, then $\Phi(p,v,t)$ is defined.

ii) If $\Phi(p,v,t)$ is defined then $\Phi(p,v,t) \in \{o,1,...,k\}$.

iii) If $\Phi(p,v,t)$ and $\Phi(p,v,t')$ are defined and $t<t'$, then $\Phi(p,v,t) \geq \Phi(p,v,t')$.

iv) If $\Phi(p,v,t)$ and $\Phi(p,v',t)$ are defined, and v is before v' on the path of p from its source to its destination, then $\Phi(p,v,t) > \Phi(p,v',t)$.

For a count-down function Φ, it means that every time a packet p is passed to a next node on its path, the Φ-value of p drops by at least one. So, for a packet p stored at v at time t, $\Phi(p,v,t)$ is an upperbound on the number of steps p has still to go to arrive at its destination node. If p is stored at v at time t and $\Phi(p,v,t)=o$, then v is the destination of p.

Important examples of count-down functions are the following.

i) Assume that a fixed routing strategy is used, i.e. for each source destination pair w_o,w_1, there is a unique path from w_o to w_1 that is followed by all packets that are send from w_o to w_1. For a packet p and a node v on the path of p, and for all t, we let F(p,v,t) denote the distance from v to the destination of p, i.e. the number of message passing steps p has to take to arrive at its destination. (F stands for "forward".)

ii) For a packet p and a node v on the path of p, let $\tilde{B}(p,v)$ denote the distance from the source of p to v, i.e. the number of message passing steps p has to take from its source to arrive at v. (Note: $\tilde{B}$ is not a count-down function.) Now let for all times t B(p,v,t) = k-$\tilde{B}(p,v)$. (B stands for "backward".)

It is easy to verify that F and B are count-down functions. Note that we always can choose B as a count-down function. (This basically follows from the assumption that each packet needs to traverse at most k links to arrive at its destination.)

Another example of a count-down function is the following. Basically B is used as the count-down function. However, as an addition, in each node v, after a packet p is stored in the buffers of v, it is tested whether one of the nodes adjacent to v is the destination of p. Is this the case, then immediately the value $\Phi(p,v,t)$ drops to 1. (Note that the "Φ-value" of p can change, even if p is not moved.)

$\Phi(p,v,t)$ can be seen as an upperbound on the number of message passing steps p has to take from v to arrive at its destination; the closer $\Phi(p,v,t)$ is to the exact number of steps p has to take, the less restrictive our resulting controllers will be. In this sense F is the "best" count-down function, and B is the "worst":

Lemma 4.2.1. Let p be a packet stored at v at time t, or let v be the next node on the path of p. Let Φ be a count-down function.

i) $\Phi(p,v,t) \leq B(p,v,t)$.

ii) Assume a fixed routing strategy is used. $\Phi(p,v,t) \geq F(p,v,t)$.

Proof.

i) v is the node of index k-B(p,v,t)+1 on the path of p. Let v_o, v_1, ..., $v_{k-B(p,v,t)}$ = v be the first k-B(p,v,t)+1 nodes on this path, and let times $t_o<t_1<_2<\ldots<t_{k-B(p,v,t)+1} = t$ be given such that $\Phi(p,v_i,t_i)$ is defined, for all i, $o\le i\le k-B(p,v,t)$. Then $k\ge\Phi(p,v_o,t_o) > \Phi(p,v_1,t_1) >\ldots> \Phi(p,v_{k-B(p,v,t)},t_{k-B(p,v,t)}) = \Phi(p,v,t)$, hence $\Phi(p,v,t) \le B(p,v,t)$.

ii) Similar. (If not a fixed routing strategy is used, then F is not defined.) □

Now we define the state of packets and of nodes. Let Φ be a count-down function. The state of packet p, stored at node v at time t can be described by the pair $(s(p),\Phi(p,v,t)) \in \{1,\ldots,q\} \times \{o,\ldots,k\}$. The state of a node v at time t is formed by the states of the packets that are stored at v at time t. We can represent this state by a q by (k+1) matrix J, with J(s,d) denoting the number of packets stored at v with state (s,d), $(1\le s\le q,\ o\le d\le k)$. J is called the statematrix of v. We use the following notions, derived from the statematrix J:

Definition. Let J be the statematrix of a processor v at certain time t.

i) The statevector of J is the vector $\vec{j} = \vec{j}(J) = <j_o,\ldots,j_{kq}>$, with $j_i = \Sigma\ J(s,d)\cdot s$, where the sum is taken over all pairs (s,d), with $s\cdot d = i$, $1\le s\le q$, $o\le d\le k$, i.e. j_i is the number of buffers used by packets with state (s,d) with $s\cdot d = i$.

ii) The number of used buffers at v is denoted by $n = \sum_{r=o}^{kq} j_r = \sum_{r_1=1}^{q} \sum_{r_2=o}^{k} (J(r_1,r_2)\cdot r_1)$.

iii) The number of free buffers at v is denoted by m = b-n.

The controllers we consider in this chapter will always allow consumption moves, but will disallow some generation and message passing moves. We consider controllers where the decision to accept a packet p at a node v at time t only depends on the pair $(s(p),\Phi(p,v,t))$, the statematrix of v, and on the global constants q, k and b. A controller, using a count-down function Φ, and constants q (for maximum packet size), k (for maximum Φ-values) and b (for buffersize) is called a uniform (Φ,q,k,b)-

controller. A network with count-down function Φ, maximum packet size q, maximum Φ-value on the network k and buffersize b is called a (Φ,q,k,b)-network. A uniform (Φ,q,k,b)-controller can be used for every (Φ,q,k,b)-network.

Formally, a uniform (Φ,q,k,b)-controller can be described as a subset $S \subseteq \{(J,(s,d)) \mid 1\leq s\leq q,\ o\leq d\leq k,\ J$ is a q by k+1 matrix with non-negative values and $\sum_{r_1=1}^{q} \sum_{r_2=o}^{k} J(r_1,r_2)\cdot r_1 \leq b-s\}$. It means that a packet p can be accepted at node v at time t, with J is the statematrix of v, if and only if $(J,(s(p),\Phi(p,v,t))) \in S$.

If we do not use the statematrix J, but statevectors $\vec{j}$ or numbers n or m, we will describe the controllers with sets of pairs $(\vec{j},(s,d))$, $(n,(s,d))$ or $(m,(s,d))$. Controllers that do not use J or $\vec{j}$, but only n or m are called "count"-controllers, the others are called "state"-controllers. In the case that q=1 we also use pairs $(\vec{j},d)$, (n,d) and (m,d).

4.3. Uniform local deadlock-free controllers. We first recall some results of Toueg and Ullman [TU81]. Consider the following 4 controllers. In each case we have q=1 and $b\geq (k+1)q = k+1$. (For i) and ii) assume that a fixed routing strategy is used.)

i) A node v with m free buffers accepts a packet p at time t iff $(m,F(p,v,t)) \in FC(b,k) = \{(m,j) \mid (j<m) \text{ and } (o\leq j\leq k) \text{ and } (1\leq m\leq b)\}$. (FC stands for "forward count".)

ii) A node v with statevector $\vec{j}$ (with respect to count-down function F) accepts a packet p at time t iff $(\vec{j},F(p,v,t)) \in FS(b,k) = \{(\vec{j},j) \mid (\forall i,\ o\leq i\leq j,\ i<b-\sum_{r=i}^{k} j_r) \text{ and } (o\leq j\leq k) \text{ and } (o \leq \sum_{r=o}^{k} j_r \leq b-1)\}$. (FS stands for "forward state".)

iii) A node v with n used buffers accepts a packet p at time t, iff $(n,B(p,v,t)) \in BC(b,k) = \{(n,i) \mid (i\geq n-b+(k+1)) \text{ and } (o\leq i\leq k) \text{ and } (o\leq n\leq b-1)\}$. (BC stands for "backward count".)

iv) Let v be a node with statevector $\vec{j}$, with respect to count-down function B. Write $\vec{i}=<i_o,\ldots,i_k>$ with $i_r=j_{k-r}$ $(o\leq r\leq k)$, so i_r denotes the number of packets, stored at v, that have made r steps so far, i.e. the distance from the source of these packets to v is r. Recall that $\tilde{B}(p,v)$ denotes the distance from the source of p to v. v accepts a packet p at time t, iff $(\vec{i},\tilde{B}(p,v)) \in BS(b,k)=\{(\vec{i},i) \mid (\forall j,\ i\le j\le k,\ j \ge \sum_{r=o}^{j} i_r -b+(k+1))$ and $(o\le i\le k)$ and $(o \le \sum_{r=o}^{k} i_r \le b-1)\}$. (BS stand for "backward state".)

Theorem 4.3.1. [TU81]

i) For $b\ge k+1$, $FC(b,k)$ and $FS(b,k)$ are deadlock-free uniform $(F,1,k,b)$-controllers.

ii) For $b\ge k+1$, $BC(b,k)$ and $BS(b,k)$ are deadlock-free uniform $(B,1,k,b)$-controllers.

A generalization of the controllers $FC(b,k)$ and $BC(b,k)$ is the following class of count-controllers:

- Let Φ be some count-down function. A node v accepts a packet p at time t, iff $(m,(s(p),\Phi(p,v,t))) \in \Phi CV(q,k,b) = \{(m,(s,d)) \mid ((d+1)s\le m)$ and $(1\le s\le q)$ and $(o\le d\le k)$ and $(1\le m\le b)\}$. $\Phi CV(q,k,b)$ is a uniform (Φ,q,k,b)-controller.

A generalization of the controllers $FS(b,k)$ and $BS(b,k)$ is the following class of state-controllers:

- Again Φ is some count-down function. A node v, with statevector $\vec{j}$ (with respect to Φ) accepts a packet p at time t, iff $(\vec{j},(s(p),\Phi(p,v,t))) \in \Phi SV(q,k,b)=\{(\vec{j},(s,d)) \mid (\forall i,\ o\le i\le ds,\ i+s\le b-\sum_{r=i}^{kq} j_r)$ and $(1\le s\le q)$ and $(o\le d\le k)$ and $(o\le \sum_{r=o}^{kq} j_r \le b-s)\}$. $\Phi SV(q,k,b)$ is a uniform (Φ,q,k,b)-controller.

If $\Phi=F$, then we write $\Phi CV(q,k,b)=FCV(q,k,b)$ and $\Phi SV(q,k,b) = FSV(q,k,b)$. Likewise for $\Phi=B$, etc. If q,k and b are clear from the context, then we write ΦCV, and ΦSV.

For uniform (Φ,q,k,b) controllers S and T we write $S \subseteq T$, iff every move that is allowed by S, is also allowed by B. If $S \subseteq T$ and $T \subseteq S$ then we write $S\equiv T$. If $S \subseteq T$ and not $S\equiv T$, then we write $S \subset T$.

Lemma 4.3.2. Let q=1, and b > k+1.

i) FC(b,k) ≡ FCV(1,k,b).

ii) BC(b,k) ≡ BCV(1,k,b).

iii) FS(b,k) ≡ FSV(1,k,b).

iv) BS(b,k) ≡ BSV(1,k,b).

Proof.

We only give a proof of iv), the other results follow directly from the definitions. Let p be a packet that wants to be accepted by node v at time t. Let $\vec{j}$ be the statevector of p (with respect to count-down function B), and let $\vec{i} = <i_o,\dots,i_k>$ be given by $i_r = b_{k-r}$ (o≤r≤k). Further write s = s(p), $i=\tilde{B}(p,v)$ = k−B(p,v,,t). Now we have:

BS(b,k) lets v accept p

$\Leftrightarrow$ (∀j, i≤j≤k, $j \geq \sum_{r=o}^{j} i_r - b+(k+1)$) and (o≤i≤k) and (o≤ $\sum_{r=o}^{k} i_r \leq b-1$)

$\Leftrightarrow$ (∀j, i≤j≤k, $j \geq \sum_{r=k-j}^{k} j_r - b+(k+1)$) and (o≤i≤k) and (o≤ $\sum_{r=o}^{k} j_r \leq b-1$)

$\Leftrightarrow$ (∀j, o≤j≤k−i, $k-j \geq \sum_{r=j}^{k} j_r - b+(k+1)$) and (o≤k−i≤k) and (o≤ $\sum_{r=o}^{k} j_r \leq$ b−1)

$\Leftrightarrow$ (∀j, o≤j≤B(p,v,t), $j+1 \leq b - \sum_{r=j}^{k} j_r$) and (o≤B(p,v,t)≤k) and (o≤ $\sum_{r=o}^{k} j_r \leq$ b−1)

$\Leftrightarrow$BSV (1,k,b) lets v accept p. □

Lemma 4.3.3. Let q,k,b be given and let Φ_1,Φ_2 be count-down functions, such that ∀p,v,t, if Φ_1(p,v,t) is defined and Φ_2(p,v,t) is defined then $\Phi_1(p,v,t) \geq \Phi_2(p,v,t)$. Then

i) Φ_1CV(q,k,b) ⊆ Φ_2CV(q,k,b).

ii) Φ_1SV(q,k,b) ⊆ Φ_2SV(q,k,b).

Proof.

i) This follows directly from the definition of ΦCV.

ii) Let v be a node. Let the statevector of v at time t with respect to Φ_1 be $\vec{j}^{\,1} = <j^1_o,j^1_1,\dots,j^1_{kq}>$, and let the statevector of v at time t with respect to Φ_2 be $\vec{j}^{\,2} = <j^2_o,j^2_1,\dots,j^2_{kq}>$. Because we have for each packet p^1, stored in v at time t, that $\Phi_1(p^1,v,t) \geq \Phi_2(p^1,v,t)$, it follows that $\sum_{r=i}^{kq} j^1_r \geq \sum_{r=i}^{kq} j^2_r$, for every i, o≤i≤kq. Now suppose Φ_1CV lets v accept packet p at time t. Then $\forall i, o \le i \le \Phi_1(p,v,t).s(p)$, $i+s(p) \le b - \sum_{r=i}^{kq} j_r^1$, so $\forall i, o \le i \le \Phi_2(p,v,t).s(p)$, $i+s(p) \le b - \sum_{r=i}^{kq} j_r^1 \le b - \sum_{r=i}^{kq} j_r^2$, hence Φ_2CV lets v accept p at time t. □

Lemma 4.3.4. For all $q \ge 1$, $k \ge 1$, $b \ge (k+1)q$, and count-down functions Φ, $\Phi CV(q,k,b) \subset \Phi SV(q,k,b)$.

Proof.

We first show that $\Phi CV \subseteq \Phi SV$. Suppose ΦCV lets a node v with statevector $\vec{j}$ at time t accept a packet p. Write $s=s(p)$, and $d=\Phi(p,v,t)$. Now $(d+1)s \le b - \sum_{j=o}^{kq} j_r$. Hence, for all i, $o \le i \le ds$, $i+s \le ds+s \le b - \sum_{j=o}^{kq} j_r \le b - \sum_{j=i}^{kq} j_r$, so ΦSV lets v accept p, at time t.

Next we show there are moves allowed by ΦSV, that are not allowed by ΦCV. Consider a node v with an empty buffer. Let v accept successively $b(v)-1$ packets with $\Phi(p,v,t)=o$ and $s(p)=1$ (this is possible with ΦCV and with ΦSV). Now try to let v accept a packet p* with $\Phi(p^*,v,t)=1$ and $s(p^*)=1$. It easily follows that ΦSV allows this move, but ΦCV does not allow this move. (There are many other examples.) □

We now give our main theorem. The proof of this theorem is similar to the proof in [TU81], that FS is deadlock-free. The introduction of the notion of count-down functions allows us to treat the "forward" and "backward" controllers as special cases of a more general notion.

Theorem 4.3.5. Let $q \ge 1$, $k \ge 1$, $b \ge (k+1)q$, and Φ a count-down function. Let S be a uniform (Φ,q,k,b)-controller with $\Phi CV(q,k,b) \subseteq S \subseteq \Phi SV(q,k,b)$. Then S is deadlock-free.

Proof.

Suppose S is not deadlock-free. Then consider a (Φ,q,k,b)-network G and suppose the network reaches a state where one or more packets are deadlocked. By making every move that is possible, we can reach a state, where every packet is deadlocked. So no packet in the network can move at a certain time t, and there is at least one packet p in the network. Let p_1 be such a packet; let v_1 be the node where p_1 is stored, and let $d_1 = \Phi(p_1,v_1,t)$, $s_1 = s(p_1)$. It is clear that v_1 is not the destination of p_1, else v_1 can consume p_1. Let v_2 be the next node on the path of p_2 to its destination, after v_1, and let $\vec{j} = \{j_o,\ldots,j_{kp}\}$ be the statevector of v_2. Now note that $\sum_{r=o}^{kq} j_r \neq o$, else p_1 is accepted in v_2. So there is at least one other packet in v_2, this packet can also not move. Let p_2 be a packet in v_2, such that $\Phi(p_2,s_2,t)s(p_2)$ is minimal over all packets in v_2:

$$\Phi(p_2,v_2,t).s(p_2) = \min\ \{r \mid j_r > o\}.$$

Let $d_2 = \Phi(p_2,v_2,t)$, $s_2 = s(p_2)$. The statevector of v_2 can be written as $\vec{j} = \langle o,\ldots,o,j_{d_2s_2},\ldots,j_{kq}\rangle$. We have that $d_2 \neq o$, else p_2 can consume v_2. Now we claim that $d_2s_2 < d_1s_1$.

Suppose p_s is the last packet that is accepted by v_2 and is stored at v_2 at time t; let t_s be the moment on which p_s was accepted by v_2. (Note that at time t_s p_s is not yet stored at v_2.) Let $\vec{j}^{\,s}$ be the statevector of v_2 at time t_s. We write $d_s = \Phi(p_s,v_2,t_s)$, $s_s = s(p_s)$. We have that $d_ss_s \geq \Phi(p_s,v_2,t)s_s \geq d_2s_2$. Note that packets stored at v_2 at t_s can have been moved out of v_2 (either by a consumption move or a passing move). Also for some packets, stored at v_2 at time t_s and time t, it can be that $\Phi(p_s,v_2,t_s) > \Phi(p_s,v_2,t)$. For all these packets $\Phi(p_s,v_2,t_s) \geq \Phi(p_s,v_2,t)$. The only packet that is newly stored in the buffers of v_2 is p_2. Note that we can ignore packets that are accepted at p_2 after t_s and have moved out of p_2 before t. Therefore one has:

$$\forall i,\ o \leq i \leq d_ss_s,\quad \sum_{r=i}^{kq} j_r^s \geq \sum_{r=i}^{kq} j_r - s_s$$

$$\forall i,\ d_ss_s < i \leq kq,\quad \sum_{r=i}^{kq} j_r^s \geq \sum_{r=i}^{kq} j_r.$$

S lets a node with statevector $\vec{j}^{\,s}$ accept a packet with state (s_s,d_s), and $S \subseteq \Phi SV(q,k,b)$, hence this move is also allowed by $\Phi SV(q,k,b)$ and therefore one has:

$$\forall i,\ o \leq i \leq d_ss_s,\ i + s_s \leq b - \sum_{r=i}^{kq} j_r^s.$$

Take $i = d_2s_2$ and one gets:

$$d_2s_2+s_s \le b - \sum_{r=d_2s_2}^{kq} j_r^s \le b - \sum_{r=d_2s_2}^{kq} j_r + s_s ,$$

hence

$$d_2s_2 \le b - \sum_{r=d_2s_2}^{kq} j_r.$$

We have also that S does not let v_2 accept p_1 at time t. Because $\Phi CV(q,k,b) \subseteq S$, this move is also not allowed by $\Phi CV(q,k,b)$. So we have that

$d_1s_1 \ge (\Phi(p_1,v_2,t)+1)\ s_1 > b - \sum_{r=o}^{kq} j_r$ (= the number of free buffers of v_2 at time t.)

Now suppose $d_2s_s \ge d_1s_1$. Then $b - \sum_{r=d_2s_2}^{kq} j_r = b - \sum_{r=o}^{kq} j_r < d_1s_1 \le d_2s_2 \le b - \sum_{r=d_2s_2}^{kq} j_r$. Contradiction. Hence $d_2s_2 < d_1s_1$.

The argument now can be repeated with p_2 instead of p_1, and so on. In this way we obtain packets p_1, p_2, p_3,..., stored at nodes v_1, v_2, v_3,... (these nodes are not necessarily all different), with $\Phi(p_1,v_1,t)s(p_1) > \Phi(p_2,v_2,t)s(p_2) > \Phi(p_3,v_3,t)s(p_3) > \ldots$ etc. This contradicts the fact that all values $\Phi(p_i,s_i,t)$ and $s(p_i)$ are non-negative. Hence S must be deadlock-free. □

Corollary 4.3.6. Let $q\ge 1$, $k\ge 1$, $b\ge(k+1)q$, and let Φ be a count-down function. $\Phi CV(q,k,b)$ and $\Phi SV(q,k,b)$ are local, uniform (Φ,q,k,b)-controllers, that are deadlock-free.

4.4. Optimality of ΦSV.

Definition. Let q,k,b and a count-down function Φ be given.

i) Φ is strict, if it is possible to use Φ as a count-down function in all networks G, for which we do not have to send messages between nodes that have a distance of more than k.

ii) Φ is stable, if for all packets p, nodes v and times t,t', if $\Phi(p,v,t)$ and $\Phi(p,v,t')$ are defined, then $\Phi(p,v,t) = \Phi(p,v,t')$.

If Φ is a strict count-down function, then we can use it in all networks where we want to send messages between each pair of nodes with a distance k between each other. An example of a count-down function, that is not strict is the function Φ, with, if $F(p,v,t)$ is defined, then $\Phi(p,v,t) = 2F(p,v,t)$. Note that, for a strict count-down function Φ, a packet p, with the distance between the source of p and the destination of p is k, and a node v on the path of p, one has $\Phi(p,v,t) = B(p,v,t)$, and if F exists (i.e. a fixed routing strategy is used), then $\Phi(p,v,t) = F(p,v,t)$.

In this section we will consider strict and stable count-down functions only. Important examples of strict and stable count-down functions are F and B. The results in this section are mainly rather straightforward generalizations of the work of Toueg and Ullman [TU81, p.598-607]. We generalize these results in two ways:

- we allow packets to have a variable size
- the results are valid for every strict and stable count-down function. (Toueg and Ullman only considered controllers, depending on F or B.)

Let integers $q \geq 1$, $k \geq 1$, $b \geq 1$ and strict, stable count-down function Φ be given. Suppose S is a local, uniform (Φ,q,k,b)-controller. For statematrices J_o and J_1 we write $J_o \underset{S}{\vdash} J_1$, iff and only if there is exactly one pair $(s,d) \in \{1,\ldots,q\} \times \{o,\ldots,k\}$ with $J_o(s,d) \neq J_1(s,d)$, and

i) for this pair (s,d) one has $J_o(s,d)-1 = J_1(s,d) \geq o$ ("a packet with state (s,d) has left the node") or

ii) for this pair (s,d) one has $J_o(s,d)+1 = J_1(s,d)$ and $\forall t$ $(J_o,(s,d,t)) \in S$, i.e. S allows a node with statematrix J_o to accept a packet with state (s,d).

The transitive closure of the relation $\underset{S}{\vdash}$ is denoted by $\underset{S}{\vdash^*}$: $J_o \underset{S}{\vdash^*} J_1$, if there are $J^o=J_o, J^1, J^2, \ldots, J^i=J_1$ with $J^o \underset{S}{\vdash} J^1$, $J^1 \underset{S}{\vdash} J^2, \ldots, J^{i-1} \underset{S}{\vdash} J^i$. Let Q be the statematrix consisting of only zero's: $Q(s,d) = o$ for all s,d, $1 \leq s \leq q$, $o \leq d \leq k$. If $Q \underset{S}{\vdash^*} J$, then we write $\underset{S}{\vdash^*} J$ (J is "syntactical reachable"). Conversely, we write $\underset{S}{\models^*} J$ (J is "network reachable"), iff there exists a network G, such that, starting from the state where all buffers of all nodes are empty, we can make a series of moves, allowed by S, such that after these moves there is a node with state J.

Lemma 4.4.1. Let Φ be a strict and stable count-down function, let $q\geq 1$, $k\geq 1$, $b\geq 1$. Let S be a local, uniform (Φ,q,k,b)-controller. Then for each statematrix J:

$$\vdash^*_S J \Leftrightarrow \models^*_S J.$$

Proof.

Suppose $\vdash^*_S J$, so there are statematrices $J^1, J^2,\ldots, J^i$, with $Q \vdash_S J^1$, $J^1 \vdash_S J^2,\ldots,J^{i-1} \vdash_S J^i$. With induction to j we will show that in the unidirectional ring with k+1 nodes R_{k+1} (see fig. 4.4.1.) there is a series of moves, allowed by S, starting from the state where all buffers are empty, such that after the moves v_o has statematrix J^j, and all other nodes have statematrix Q. Write $J^o = Q$. It is clear, that the induction hypothesis holds for j=o. Now suppose the induction hypothesis holds for certain j. Then we can make a series of moves, starting from the state where all buffers of all nodes are empty, resulting in the state where v_o has statematrix J^j, and all buffers of all other nodes empty. We have that $J^i \vdash_S J^{j+1}$. If there is a pair (s,d) with $J^j(s,d) - 1 = J^{j+1}(s,d) \geq o$, then we can move a packet with state (s,d) to its destination and consume it there. If there is a pair (s,d) with $J^i(s,d) + 1 = J^{j+1}(s,d)$, and $\forall t, (J^j,(s,d,t)) \in S$, then create a packet p with size s in v_{d+1}, with destination v_d. This packet can be moved to v_o, and, because $(J^j,(s,d)) \in S$, it can be accepted in v_o. (Note that $\forall t$ $\Phi(v_o,p,t) = B(v_o,p,t) = d$.) Hence the induction hypothesis holds also for j+1. We conclude that $\models^*_S J$.

Now suppose that $\vdash^*_S J$. From the stableness of Φ it follows that the state of a node v can only change if a packet is accepted in v (by a passing move or a generation move), or if a packet leaves v (by a passing move or a consumption move). Both types of changes correspond to a statetransition $J^o \vdash_S J^1$. With induction one can conclude $\vdash^*_S J$. □

States J with not$(\vdash_S J)$ ($\Leftrightarrow$ not$(\models_S J)$) are called unreachable states. Without loss of generality, we can assume that considered controllers do not allow moves from unreachable states. We are now ready to give the main result in this section.

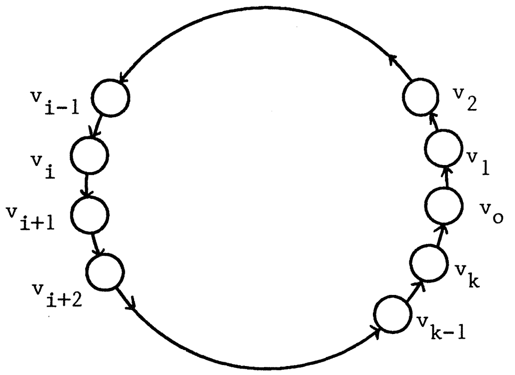


fig. 4.4.1. The unidirectional ring R_{k+1}.

<u>Theorem 4.4.2</u>. Let $q\geq 1$, $k\geq 1$, $b\geq 1$, let Φ be a strict and stable count-down function. Let S be a local, uniform (Φ,q,k,b)-controller, that does not allow moves from unreachable states.

i) If $b<(k+1)q$, then S is not deadlock-free.

ii) If $b\geq(k+1)q$ and S is deadlock-free, then $S \subseteq \Phi SV(q,k,b)$.

<u>Proof</u>.

Let q, k, b, Φ and S be given. Suppose that S is deadlock-free and $b<(k+1)q$ or not$(S \subseteq \Phi SV(q,k,b))$. We first claim there is a pair $(s,d) \in \{1,\dots,q\} \times \{o,\dots,k\}$ and a statematrix J, such that $(J,(s,d)) \in S$ and for $\vec{j} = \vec{j}(J)$:

$$\exists\ i_o,\ o\leq i_o\leq sd,\ i_o+s > b-\sum_{r=i_o}^{kq} j_r.$$

If $b<(k+1)q$, then we can choose $\vec{j} = \langle o,\dots,o\rangle$, d=k, s=q and i_o = kq. If not$(S \subseteq \Phi SV(q,k,b))$, then the result follows directly from the definition of $\Phi SV(q,k,b)$. When v accepts a packet with size s, v must have at least s free buffers, so $i_o+s > b-\sum_{r=i_o}^{kq} j_r \geq b-\sum_{r=o}^{kq} j_r \geq s$, hence $i_o \geq 1$.

Consider all pairs (s,d), such that there exists a statematrix J, with $(J,(s,d)) \in S$, and for $\vec{j}=\vec{j}(J)$ one has that

$$\exists\, i_o,\ o\leq i_o\leq sd,\ i_o+s > b-\sum_{r=i_o}^{kq} j_r.$$

Let (s_o,d_o) be such a pair, such that $s_o d_o$ is minimal over all these pairs. Let i_o be given, such that $o\leq i_o\leq s_o d_o$, $i_o+s > b-\sum_{r=i_o}^{kq} j_r$, and let J_o be the corresponding statematrix. As noted before, $i_o\geq 1$, so $s_o d_o\geq 1$, hence $d_o\geq 1$. Consider the graph G, given in fig. 4.4.2.

In the network G we will only send messages between nodes that have a distance k, i.e. we send messages from nodes v_i to nodes v_{i-1}, nodes v_i to nodes w_{i-2}, etc. We now give a series of moves, permitted by S, starting from the state where all buffers are empty, such that a deadlock will occur in G. We only describe the moves made by the nodes v_i $(o\leq i\leq k)$; the same moves are made by the nodes w_i $(o\leq i\leq k)$.

1. As in the proof of lemma 4.4.1. we can reach state J_o in v_o, with messages with source and destination in $\{v_i | o\leq i\leq k\}$. Let $\vec{j}=\vec{j}(J_o)$.
2. Generate a packet p with size s_o, in node $v_{(d_o+1)\mathrm{mod}(k+1)}$ with

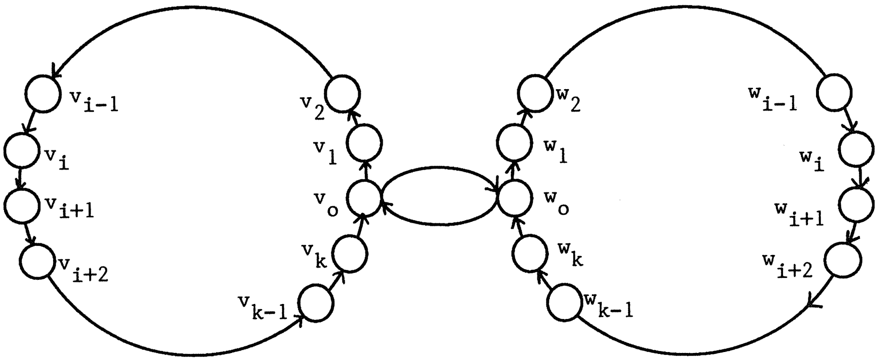


fig. 4.4.2.

destination $w_{(d_o-1)}$. Note that $\forall t$, $\Phi(p,v_o,t) = B(p,v_o,t) = d_o$. Pass p along the successive nodes on the ring $\{v_i | o \le i \le k\}$, and accept it in v_o (this is possible, because $(J_o,(s_o,d_o)) \in S$). Let J_1 be the statematrix of v_o, just after p is placed in the buffers of v_o, and write $\vec{j}^{\,1} = \vec{j}(J^1)$. We have that

$$j^1_r = j^o_r \text{ , if } r \ne s_o d_o \text{, and}$$

$$j^1_{s_o d_o} = j^o_{s_o d_o} + s_o .$$

3. Let all packets p, in v_o, with $\Phi(p,v_o,t).s(p) \le i_o$ be passed to their destination, and consumed there. Let the resulting statematrix of v_o be J_2, and write $\vec{j}^{\,2} = \vec{j}(J_2)$. Now we have

$$j^2_r = o \text{, if } o \le r \le i_o - 1 \text{ , and}$$

$$j^2_r = j^1_r = j^o_r \text{ , if } i_o \le r \le kq \text{ and } r \ne s_o d_o \text{, and}$$

$$j^2_{s_o d_o} = j^1_{s_o d_o} = j^o_{s_o d_o} + s_o .$$

4. The destination of the packets that still are in v_o is changed, such that the next node on their path is w_o. (So a packet p with $\Phi(p,v_o,t) = d$, now has destination w_{d-1}.) Note that for each packet p in v_o one has $\forall t$ $B(p,v_o,t) = F(p,v_o,t) = \Phi(p,v_o,t) \ge i_o \ge 1$.

5. As long as it is possible to accept in v_o packets p with $s(p).\Phi(p,v_o,t) \ge s_o d_o$ (with regard to S), such packets p are generated in $v_{(\Phi(p,v_o,t)+1) \bmod (k+1)}$ with destination $w_{\Phi(p,v_o,t)-1}$. The packets are passed along the successive nodes on the ring $\{v_i | o \le i \le k\}$, and are accepted by v_o. We stop iff this is no longer possible, i.e. v_o reaches a state(matrix) J_3, with for all s,d; $o \le d \le k$, $1 \le s \le q$, $sd \ge s_o d_o$, $(J_3,(s,d)) \notin S$. Write $\vec{j}^{\,3} = \vec{j}(J_3)$. We have that

$$j^3_r = o \text{ , iff } o \le r \le i_o - 1 \text{ , and}$$

$$j^3_r \ge j^2_r = j^o_r \text{ , iff } i_o \le r \le kq \text{ and } r \ne s_o d_o \text{, and}$$

$$j^3_{s_o d_o} \ge j^2_{s_o d_o} = j^o_{s_o d_o} + s_o .$$

Now G is deadlocked. There is at least one packet in v_o ($j^3_{s_o d_o} \ge s_o$), and every packet in v_o is directed towards w_o. The statematrix of w_o is also J_3. We now claim $\forall$ s,d, $1 \le s \le q$, $o \le d \le k$, $sd \ge s_o d_o$: $(J_3,(s,d-1)) \notin S$.

Suppose there are s,d, $1 \le s \le q$, $o \le d \le k$, $sd \ge s_o d_o$ and $(J_3,(s,d-1)) \in S$. First note that $(d-1)s < s_o d_o$ (because of the construction of J_3). Further note that $b- \sum_{r=i_o-s}^{kq} j_r^3 \le b- (j_{i_o}^o + \ldots + (j_{s_o d_o}^o + s_o) + \ldots + j_{kq}^o) =$

$(b- \sum_{r=i_o}^{kq} j_r^o) - s_o < (i_o+s_o) - s_o = i_o$. So $\exists\, i_1$, $o \le i_1 \le (d-1)s$, $i_1+s >$ $b- \sum_{r=i_1}^{kq} j_r^3$ (Choose $i_1 = i_o - s$, so $i_1 \le s_o d_o - s \le (d-1)s$). From the definition of d_o and s_o, and from $(d-1)s < s_o d_o$, we now have that $(J_3,(s,d-1)) \notin S$. Contradiction.

This means that none of the packets that is stored at v_o can be accepted in w_o. Similarly none of the packets that is stored at w_o can be accepted in v_o. So G is deadlocked. This shows that our initial assumption that S is deadlock-free and $b<(k+1)q$ or not$(S \subseteq \Phi SV(q,k,b))$ is false. □

For q=1 we have that for every deadlock-free, local, uniform (Φ,q,k,b) <u>count</u>-controller S (with q=1, $k \ge 1$, $b \ge 1$, Φ a strict and stable count-down function) $S \subseteq \Phi CV(1,k,b)$. (The proof is similar to the proof in [TU81], for the case that Φ=F.) For q>1 this result does not hold:

<u>Proposition 4.4.3</u>.

i) Let q>1, $k \ge 1$, $b \ge (q+1)k$, and let Φ be a count-down function. S = $\{ (m,(s,d)) \mid (((d+1)s \le m)$ or $((b-m)k \le m-s))$ and $(1 \le s \le q)$ and $(o \le d \le k)$ and $(s \le m \le b) \}$ is a deadlock-free, local, uniform (Φ,q,k,b)-controller.

ii) If $b \le (k+1)q+q-2$, then $\Phi CV(q,k,b) \subset S$

iii) If $b > (k+1)q+q-2$, then $\Phi CV(q,k,b) \equiv S$

<u>Proof</u>.

i) Let q,k,b,Φ be given. It is clear that $\Phi CV(q,k,b) \subseteq S$. If $S \subseteq \Phi SV(b,k,q)$, then we have by theorem 4.4.2. that S is deadlock-free.

Suppose that not$(S \subseteq \Phi SV)$. Then there are s,d and a statematrix J, $(1 \le s \le q$, $o \le d \le k)$, with $(J,(s,d)) \in S$, and for $\vec{j} = \vec{j}(J)$, $\exists\, i_o$, $o \le i_o \le sd$, $i_o+s > b- \sum_{r=i_o}^{kq} j_r$ and $k.\sum_{j=o}^{kq} j_r < b - \sum_{r=o}^{kq} j_r - s$. Let s,d,J and i_o be given. We have that $(d+1)s \ge i_o + s > b - \sum_{r=i_o}^{kq} j_r \ge b - \sum_{r=o}^{kq} j_r \ge k.\sum_{r=o}^{kq} j_r + s$, hence $sd > k.\sum_{r=o}^{kq} j_r$. From $d \le k$, it follows that $q \ge s > \sum_{r=o}^{kq} j_r$. Now we consider two cases.

CASE I : $i_o > k(q-1)$. From $\sum_{r=o}^{kq} j_r < q$, it follows that at a node with statematrix J, there are no packets stored with size q, hence for all $r \ge k(q-1)+1$ one has $j_r = o$. So $\sum_{r=i_o}^{kq} j_r = o$. Now $kq \ge sd \ge i_o \ge b-s \ge (k+1)q-s \ge kq$. Contradiction.

CASE II : $i_o \le k(q-1)$. Now $kq \ge i_o + q \ge i_o + s > b - \sum_{r=i_o}^{kq} j_r \ge b - \sum_{r=o}^{kq} j_r > b-q \ge kq$. Contradiction.

We conclude that $S \subseteq \Phi SV(q,k,b)$, and hence that S is deadlock-free.

ii) Let $b \le (k+1)q+q-2$. It is clear that $\Phi CV(q,k,b) \subseteq S$. We now show that not$(\Phi CV(q,k,b) \supseteq S)$. Controllers $\Phi CV(q,k,b)$ and S allow a node v with all buffers empty to accept a packet with size $q-1$. Now try to accept a packet with state (q,k). Notice that $(k+1)q > b-(q-1) = m$, hence $(m,(q,k)) \notin \Phi SV(q,k,b)$ and $(b-m)k = (q-1)k \le b-3q+2 \le m-q$, hence $(m,(q,k)) \in S$. S allows a node with $b-(q-1)$ free buffers to accept a packet with state (q,k), $\Phi CV(q,k,b)$ does not.

iii) Let $b > (k+1)q+q-2$. Again it is clear that $\Phi CV(q,k,b) \subseteq S$. We now show that $\Phi CV(q,k,b) \supseteq S$. Suppose $(m,(\Phi,s)) \notin \Phi CV(q,k,b)$ and $(m,(\Phi,s)) \in S$, for some m,Φ,s, $o \le m \le b$, $o \le \Phi \le k$, $1 \le s \le q$. Then $(k+1)s > m$ and $(b-m)k+s \le m$. It follows that $(k+1)s > (b-m)k+s$, hence $s > b-m$ and $q > b-m$. Now $(k+1)q \ge (k+1)s > m > b-q$, hence $b \le (k+1)q+q-2$, contradiction. □

It is presently open to find an "optimal" deadlock-free, local, uniform (Φ,q,k,b)-count-controller for $q>1$.

4.5. Final comments.

i) It is possible to vary the number of buffers at each node. The results of this chapter can easily be generalized for the case that the condition $b \geq (k+1)q$ is replaced by the condition: for each node v, the number of buffers of v b(v) is at least $\max\{(\Phi(p,v,t)+1)s(p) \mid p$ is a packet, and v is possibly on the path of $p\}$.

ii) The controllers ΦCV and ΦSV we presented in this chapter have an important drawback: they do not prevent lifelock. Lifelock occurs when there are packets that will never reach their destination. An example of a network with lifelock is given in fig. 4.5.1. Assume $k \geq 2$, $q \geq 3$, $b \geq (k+1)q$ to be given, and suppose the controller FSV(b,k,q) is used.

In v_o b–1 packets with destination v_2 and size 1 are created, and accepted in v_1. Now a packet p_1 with size 3 is created in w_o, with destination w_2. w_o cannot be accepted in v_o. Now one packet in v_1 passes to v_2 and is consumed there; still w_o cannot be accepted in v_o. Now a new packet with size 1 and destination v_2 is created in v_o and is accepted in v_2. Again a packet in v_1 passes to v_2, etc. So there is an infinite

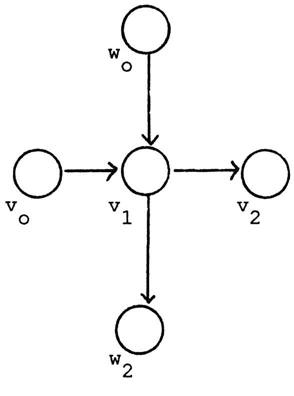


fig. 4.5.1.

sequence of moves, that prevents p_1 to move from w_o to v_1. Hence packet p_1 is lifelocked.

Toueg [To80] showed that FS(b,k) and BS(b,k) can be modified, such that the resulting controllers are lifelock free too. We will give a short informal description of the modification. The state of a packet now consists of the value F(p) (or B(p)) and the time that it was created, that is: the first moment that a processor tried to let the packet enter the buffers of a node v. (The size of the packet is 1.) A processor v accepts a packet p, if and only if the move is allowed according to controller FS(b,k) (or BS(b,k)), and there is no other packet p′ waiting to be accepted by v, with an earlier creating time, for which the acceptance of p′ by v is also allowed by FS(b,k) (or BS(b,k)). These controllers are called FSET(b,k) and BSET(b,k), respectively. The same modification can also be applied if we use other count-down functions than F or B. However, the same modifications to controllers ΦCV(q,k,b) or ΦSV(q,k,b) with q>1 do not yield lifelock-free controllers. A straightforward, but not very interesting way to find lifelock- and deadlock-free controllers that allow the packet size to vary, is to treat each packet as if its size is q and then use basically FSET($\lfloor \frac{b}{q} \rfloor$,k) or BSET($\lfloor \frac{b}{q} \rfloor$,k). (Packets are treated as if they all have the same size.) It remains an interesting and challenging open problem to find lifelock and deadlock-free controllers, that make an (essential) use of the differences in packet sizes.

CHAPTER FIVE

EMULATIONS: DEFINITIONS AND CHARACTERIZATIONS

5.1. Introduction. Parallel algorithms are normally designed for execution on a suitable network of N processors (viewed as SIMD- or MIMD-machine [vL85]), with N depending on the size of the problem to be solved. In practice N will be large and varying, whereas processor networks will be small and fixed. The resulting disparity between algorithm design and implementation must be resolved by simulating a network of some size N on a fixed and smaller size network of a similar or different kind, in a structure preserving and efficient manner. Notions of simulation are well-understood in e.g. automata theory (see [He71]), and suitable analogs can be brought to bear on networks of processors. In this chapter and in chapter 6 and 7 we study a notion of simulation, termed: emulation, that was recently proposed by Fishburn and Finkel [FF82]. Independently Berman [Be83] proposed a similar notion.

Definition. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be networks of processors (graphs). We say that G can be emulated on H if there exists a function $f: V_G \rightarrow V_H$ such that for every edge $(g,g') \in E_G$: $f(g) = f(g')$ or $(f(g), f(g')) \in E_H$. The function f is called an emulation function, or in short, an emulation of G on H.

Clearly, emulation between networks is transitive. We shall only be interested in emulations f that are "onto".

Let f be an emulation of G on H. Any processor $h \in V_H$ must actively emulate the processors $\in f^{-1}(h)$ in G. When $g \in f^{-1}(h)$ communicates information to a neighboring processor g', then h must communicate the corresponding information "internally" when it emulates g' itself or to a neighboring processor $h' = f(g')$ in H otherwise. If all processors act synchronously in G, then the emulation will be slowed by a factor proportional to $\max_{h \in V_H} |f^{-1}(h)|$.

Definition. Let G, H, and f be as above. The emulation f is said to be (computationally) uniform if for all h, h' $\in V_H$: $|f^{-1}(h)| = |f^{-1}(h')|$. f is uniform iff $cc(f) = |V_G|/|V_H|$.

Every uniform emulation f has associated with it a fixed constant c, called: the computation factor, such that for all h $\in V_H$: $|f^{-1}(h)| = c$. It means that every processor of H emulates the same number of processors of G. Again, uniform emulation between networks is transitive. When G can be uniformly emulated on H and H can be uniformly emulated on G, then G and H are necessarily isomorphic. (Thus uniform emulation establishes a partial ordering of networks.) A number of useful, elementary results on (uniform) emulations are given in section 5.2.

The relevant question is whether (large) networks of some class C can be uniformly emulated by networks of a smaller size within the same class C. Fishburn and Finkel [FF82] answered this question affirmatively for the following classes of processor networks : the shuffle-exchange network, the grid-connected network, the n-dimensional cube, the plus-minus network, the binary lens, and the cube-connected cycles. (For definitions of these networks, see [FF82].) In this chapter we shall take a more fundamental approach and develop a detailed analysis of all possible emulations in selected classes of networks : the shuffle-exchange network, the n-dimensional cube, the ring, and the 2-dimensional grid. The results will be presented in sections 5.3. to 5.5. Some proofs are deferred to appendix B and C. In chapters 6 and 7 we proceed with the analysis of (uniform) emulations. In chapter 6 we give a further analysis of (uniform) emulations. Chapter 7 is devoted to the complexity of the problem to find uniform emulations between networks of certain characteristics, and related questions.

5.2. Elementary results. In this section we present some elementary results on (uniform) emulations.

Lemma 5.2.1. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be connected graphs (or strongly connected directed graphs). $f : V_G \to V_H$ is an emulation, if and only if for all $v,w \in V_G$: $d_G(v,w) \geq d_H(f(v),f(w))$.

Proof.

$\Rightarrow$Use induction to $d_G(v,w)$.

$\Leftarrow$Trivial. □

Recall the definitions of Cartesian sum, Cartesian product, normal product and composition of graphs from section 1.5.

Lemma 5.2.2. G can be uniformly emulated on H if and only if there exists a graph G′ such that G is a spanning subgraph of H[G′].

Proof.

$\Rightarrow$Let f be a uniform emulation of G on H with computation factor c. The sets $\{f^{-1}(h)\}$, $h \in H$, partition G into blocks of size c. Let G′ be any graph on c nodes such that the induced subgraph of every block (in G) is contained in G′. Next observe that for any two nodes $g \in f^{-1}(h)$ and $g' \in f^{-1}(h')$ of G : $(g,g') \in E_G \Rightarrow h = h'$ (and the edge is in G′) or $(h,h') \in E_H$. It follows that G is a spanning subgraph of H[G′].

$\Leftarrow$From the definition of composition (cf. [Ha69]), by projection on H. □

G′ can always be chosen to be equal to K_c, the complete graph on c nodes. When G is uniformly emulated on H, then H can be viewed as a "factor" of G (and, in particular: $|V_H|\,|\,|V_G|$). When $|V_G|=|V_H|$, then G can be uniformly emulated on H if and only if G is isomorphic to a subgraph of H.

Lemma 5.2.3. Let $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$, $H_1=(W_1,F_1)$, $H_2=(W_2,F_2)$ be graphs; and let f_1: $V_1 \to W_1$ and f_2: $V_2 \to W_2$ be (uniform) emulations of G_1 on H_1 and G_2 on H_2, respectively. Let $f_1 \times f_2$ denote the mapping $V_1 \times V_2 \to W_1 \times W_2$, given by $f_1 \times f_2((v_1,v_2)) = (f_1(v_1),f_2(v_2))$, for all $v_1 \in V_1$, $v_2 \in V_2$. Then:

(i) $f_1 \times f_2$ is a (uniform) emulation of G_1+G_2 on H_1+H_2.

(ii) $f_1 \times f_2$ is a (uniform) emulation of $G_1 \bullet G_2$ on $H_1 \bullet H_2$.

Proof. The results follow easily from the definitions. □

For the Cartesian product and the composition, results similar to lemma 5.2.3. do not hold. Consider $G_1 = P_4$, $H_1 = P_2$, $G_2 = H_2 = P_3$. Let the functions f_1: $\{0,1,2,3\} \rightarrow \{0,1\}$ and f_2: $\{0,1,2\} \rightarrow \{0,1,2\}$ be given by $f_1(0)=f_1(1)=0$ and $f_1(2)=f_1(3)=1$; and $f_2(i)=i$ for $i=0,1,2$. It is obvious that f_1 is a uniform emulation of P_4 on P_2 and f_2 is a uniform emulation of P_3 on P_3. However, $f_1 \times f_2$ is *not* an emulation of $G_1 \times H_1 = P_4 \times P_3$ on $G_2 \times H_2 = P_2 \times P_3$ (use that $((0,0),(1,0)) \notin E_{P_4 \times P_3}$ and $(f_1 \times f_2((0,0)), f_1 \times f_2((1,2))) = ((0,0),(0,2)) \notin E_{P_2 \times P_3}$), and $f_1 \times f_2$ is *not* an emulation of $G_1[H_1] = P_4[P_3]$ on $G_2[H_2] = P_2[P_3]$ (use that $((0,0),(1,0)) \notin E_{P_4[P_3]}$ and $(f_1 \times f_2((0,0)), f_1 \times f_2((1,2))) = ((0,0),(0,2)) \notin E_{P_2[P_3]}$).

The following lemma's relate the (cyclic) bandwidth of a graph and its capacity to be uniformly emulated on a path or a ring. (Cf. section 1.5.)

Lemma 5.2.4. Let $G = (V,E)$ be an undirected graph and $K \mid |V|$. Let $n = |V|/K$. Then:

$$\begin{aligned} &\text{Bandwidth } (G) \leq K \\ &\Rightarrow G \text{ can be uniformly emulated on } P_n \\ &\Rightarrow \text{Bandwidth } (G) \leq 2K-1. \end{aligned}$$

Proof. (i) Let f be a linear ordering of G with bandwidth $\leq K$, $K \mid |V|$ and $n = |V|/K$. Then g: $V \rightarrow V_n$, defined by $g(v) = \lfloor f(v)/K \rfloor$ is a uniform emulation of G on P_n : if $(u,v) \in E$ then $|f(u)-f(v)| \leq K$, hence $|g(u)-g(v)| \leq 1$, and $g(u)$ and $g(v)$ are adjacent in P_n.

(ii) Let g be a uniform emulation of G on P_n. For every $i \in V_n$ we can number the nodes in $g^{-1}(i)$ from o to K-1. Let nb(v) be the number given to v, i.e. $\forall\, v,w \in V \; v \neq w \wedge g(v)=g(w) \Rightarrow nb(v) \neq nb(w)$ and $\forall v \in V$

$nb(v) \in \{o,..,K-1\}$. Now $f: V \to V_{|V|}$, defined by $f(v) = K \cdot g(v) + nb(v)$ is a linear ordering with bandwidth at most 2K-1. □

Lemma 5.2.5. Let $G=(V,E)$ be an undirected graph and $K \mid |V|$. Let $n = |V|/K$. Then

Cyclic Bandwidth (G) $\leq$ K
$\Rightarrow$G can be uniformly emulated on R_n
$\Rightarrow$Cyclic Bandwidth (G) $\leq$ 2K-1.

Proof. Similar to 5.2.4. □

Lemma 5.2.6. Let $G=(V,E)$ be a directed, acyclic graph. Let $K \mid |V|$ and $n = |V|/K$. Then

Dir Bandwidth (G) $\leq$ K
$\Rightarrow$G can be uniformly emulated on $\vec{P}_n$
$\Rightarrow$Dir Bandwidth (G) $\leq$ 2K-1.

Proof. (i) Similar to 5.2.4.

(ii) Let g be a uniform emulation of G on $\vec{P}_n$. For every $i \in V_n$ we can number the nodes in $g^{-1}(i)$ from o to K-1, such that if $(u,v) \in E$, $u,v \in g^{-1}(i)$, then $nb(u) < nb(v)$. Now $f: V \to V_{|V|}$, defined by $f(v) = K \cdot g(v) + nb(v)$ is a linear ordering with directed bandwidth at most 2K-1. □

Lemma 5.2.7. Let $G=(V,E)$ be a directed graph, let $K \mid |V|$ and $n = |V|/K$. Then

(i) Dir Cyclic Bandwidth (G) $\leq$ K
$\Rightarrow$G can be uniformly emulated on R_n and G does not contain a cycle with length less than n.

(ii) G can be uniformly emulated on R_n and G does not contain a cycle with length K or less
$\Rightarrow$Dir Cyclic Bandwidth (G) $\leq$ 2K-1.

Proof. (i) Let f be a linear ordering of V with directed cyclic bandwidth at most K. A uniform emulation of G on $\vec{R}_n$ can be constructed similar to 5.2.4. Suppose G contains a cycle with length less than n, with nodes $v_o, \ldots v_{l-1}$ $(l < n)$.

Then $\sum_{i=0}^{l-1} d_{\vec{R}_{|V|}}(\, f(v_i),\ f(v_{(i+1) \bmod l}\,)) = n$, so there exists a i, $o \le i \le l-1$ such that $d_{\vec{R}_{|V|}}(f(v_i),\ f(v_{(i+1) \bmod l})) \ge |V|/l > |V|/n = K$: the directed cyclic bandwidth of f is greater than K. Contradiction.

(ii) Let g be a uniform emulation of G on $\vec{R}_n$, and suppose G does not contain cycles with length K or less. For every i, $o \le i \le n-1$ look at the subgraph of G, with nodes $g^{-1}(i)$ and edges $\{(v,w) \mid v,w \in g^{-1}(i)$ and $(v,w) \in E\}$. This subgraph is cycle-free (there are K nodes in this subgraph). So we can number the nodes in $g^{-1}(i)$ from o to K-1, such that each node in $g^{-1}(i)$ has a unique number $\in \{o,\ldots,K-1\}$, and $u,v \in g^{-1}(i) \wedge (u,v) \in E \Rightarrow nb(u) < nb(v)$. This we do for every i, $o \le i \le n-1$. Now again $f: V \to V_{|V|}$, defined by $f(v) = K \cdot g(v) + nb(v)$ is a linear ordering with bandwidth at most 2K-1. □

The computation factor of the uniform emulations mentioned in lemmas 5.2.4.-5.2.7. is always K. Another useful property is the following. Recall that G^R is the (directed) graph obtained by reversing the direction of the edges of G (cf. section 1.5.).

<u>Lemma 5.2.8</u>. f is a (uniform) emulation of G on H if and only if f is a (uniform) emulation of G^R on H^R.

<u>Proof</u>.

$\Rightarrow$ Let f be an emulation of G on H. It means that for every edge $(g,g') \in E_G$: $f(g) = f(g')$ or $(f(g),\ f(g')) \in E_H$. Thus, by simple translation, we have for every edge $(g',g) \in E_{G^R}$: $f(g') = f(g)$ or $(f(g'),f(g)) \in E_{H^R}$. Hence f is an emulation of G^R on H^R.

$\Leftarrow$ By a similar argument, observing that $(G^R)^R = G$ for all graphs G. Finally we note that uniformity is preserved in the constructions. □

5.3. <u>Emulations of the 4-pin shuffle</u>. In this section we develop a detailed analysis of the possible emulations of the 4-pin shuffle (cf. section 1.6.3.). Our main result will be that there are exactly 6 different uniform emulations of S_n on S_{n-1}. We also show that there are at least $2 \cdot 2^{2^k} - 2^{2^{k-1}}$ uniform emulations of S_n on S_{n-k} $(k \geq 1)$. In section 5.3.1. we give some preliminary definitions and results, in section 5.3.2. we give the analysis leading to our main result. The proof of the main theorem is deferred to appendix B. In section 5.3.3. we discuss the uniform emulation of S_n on S_{n-k} in general and argue that the results hold for the uniform emulation of the inverse 4-pin shuffle as well.

5.3.1. Preliminaries. In this section we use the notations and definitions, introduced in section 1.5. Recall the definition of the 4-pin shuffle from section 1.6.3.

Assume $n>2$. The fact that S_n can be (uniformly) emulated on S_{n-1} and, hence, on every S_{n-k} $(k \geq 1)$ derives from the following observation, using lemma 5.2.2. (Compare [FF82], theorem 2.)

Lemma 5.3.1.1. S_n is a spanning subgraph of $S_{n-1}[\overline{K}_2]$, for $n \geq 2$.

Proof.

Consider the mapping $h : S_n \rightarrow S_{n-1}[\overline{K}_2]$ defined by $h(b_1..b_n) = \langle b_1..b_{n-1}, b_n \rangle$, which clearly is 1-1 and onto on the set of nodes. Let $(b,c) \in E_n$ with $c = b_2..b_n \frac{0}{1}$ (necessarily). Then $(b_1..b_{n-1}, b_2..b_n) \in E_n$, hence $h(b)$ and $h(c)$ are adjacent in $S_{n-1}[\overline{K}_2]$. Thus S_n is isomorphic to a spanning subgraph of $S_{n-1}[\overline{K}_2]$. □

Corollary 5.3.1.2. S_n is a spanning subgraph of $S_{n-k}[\overline{K}_{2^k}]$, for every $1 \leq k < n$.

By using a mapping h defined by $h(b_1..b_n) = \langle b_{n-1}..b_1, b_n \rangle$, a similar argument shows that S_n is a spanning subgraph of $\tilde{S}_{n-1}[K_2]$ and (hence) that S_n can be uniformly emulated on $\tilde{S}_{n-1}$ and any smaller inverse 4-pin shuffle. Clearly $\tilde{S}_n \cong S_n$.

Lemma 5.3.1.3. f is an emulation of S_n on S_{n-k} if and only if for all x

$\in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-k-1}$ and $\alpha,\beta \in (\frac{o}{1})$: if $f(\alpha x) = \beta y$ then $(f(xo) = \beta y \vee f(xo) = y\frac{o}{1})$ and $(f(x1) = \beta y \vee f(x1) = y\frac{o}{1})$.

(The proof follows straight from the definitions involved.) For a mapping f, define its "companion" $\bar{f}$ by $\bar{f}_i(b) = \overline{f_i(b)}$ for all $1 \leq i \leq n$.

Lemma 5.3.1.4. If f is an emulation of S_n on S_{n-k}, then so is $\bar{f}$.

Proof.

Immediate from lemma 5.3.1.3. □

5.3.2. Uniform emulations of S_n on S_{n-1}. The uniform emulations of S_n on S_{n-1} will be shown to be "step-simulating" in a very precise sense. Our aim will be to characterize all step-simulations of S_n on S_{n-1}, and to derive from it all uniform emulations.

Definition. A mapping $g : S_n \to S_{n-1}$ is called step-simulating (or : a "step-simulation" of S_n on S_{n-1}) if and only if for all $x \in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-2}$ and $\alpha,\beta \in (\frac{o}{1})$: if $g(\alpha x) = \beta y$, then $g(xo) = y\frac{o}{1}$ and $g(x1) = y\frac{o}{1}$.

Lemma 5.3.2.1. Every step-simulation g of S_n on S_{n-1} is an emulation.

Proof.

Immediate. (Compare lemma 5.3.1.3.) □

We shall call a step-simulation "uniform" when it is uniform as an emulation. When g is a step-simulation, then so is $\bar{g}$.

Lemma 5.3.2.2. A mapping $g : S_n \to S_{n-1}$ is step-simulating if and only if for all $x \in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-2}$ and $\alpha,\beta \in (\frac{o}{1})$: if $g(x\alpha) = y\beta$ then $g(ox) = \frac{o}{1}y$ and $g(1x) = \frac{o}{1}y$.

Proof.

By verifying equivalence with the definition of step-simulation. (Use the string character of x and y.) □

Lemma 5.3.2.2. can be interpreted as stating that the (uniform) step-simulations of S_n on S_{n-1} act at the same time as (uniform) step-simulations of $\tilde{S}_n$ on $\tilde{S}_{n-1}$. Note the following useful properties of step-simulations g:

$$g(b_1..b_{n-1}o)|_{n-2} = {}_2|g(ob_1..b_{n-1})$$
$$g(b_1..b_{n-1}1)|_{n-2} = {}_2|g(1b_1..b_{n-1})$$

We shall now aim for a characterization of the possible step-simulations and uniform step-simulations of S_n on S_{n-1}.

Definition. For n≥3, define the operators $\Pi^n : [V_n \rightarrow V_{n-1}] \rightarrow [V_{n-1} \rightarrow V_{n-2}]$ and $T^n : [V_{n-1} \rightarrow V_{n-2}] \rightarrow [V_n \rightarrow V_{n-1}]$ as follows:

$$\Pi^n(g)\ (b_1..b_{n-1}) = g(b_1..b_{n-1}o)|_{n-2}$$
$$T^n(h)\ (b_1..b_n) = h(b_1..b_{n-1}).h_{n-2}(b_2..b_n)$$

Theorem 5.3.2.3. For n≥3,

(i) if g is a step-simulation of S_n on S_{n-1}, then $\Pi^n(g)$ is a step-simulation of S_{n-1} on S_{n-2}.

(ii) if h is a step-simulation of S_{n-1} on S_{n-2}, then $T^n(h)$ is a step-simulation of S_n on S_{n-1}.

(iii) restricted to step-simulations, Π^n and T^n are inverses.

(iv) restricted to step-simulations, Π^n preserves uniformity.

Proof.

(i) Verify the condition of lemma 5.3.2.3. : $\Pi^n(g)(x\alpha) = y\beta \Rightarrow g(x\alpha o) = y\beta\frac{o}{1}$ (definition of Π^n) $\Rightarrow g(ox\alpha) = \frac{o}{1}y\beta$ and $g(1x\alpha) = \frac{o}{1}y\beta \Rightarrow g(oxo) = \frac{o}{1}y\frac{o}{1}$ and $g(1xo) = \frac{o}{1}y\frac{o}{1}$ (by shifting right and then left) $\Rightarrow \Pi^n(g)\ (ox) = \frac{o}{1}y$ and $\Pi^n(g)\ (1x) = \frac{o}{1}y$.

(ii) Similarly $T^n(h)\ (x\gamma a) = y\delta\beta \Rightarrow h(x\gamma) = y\delta \Rightarrow T^n(h)(ox\gamma) = h(ox).h_{n-2}\ (x\gamma) = \frac{o}{1}y\delta$ and $T^n(h)\ (1x\gamma) = h(1x).h_{n-2}\ (x\gamma) = \frac{o}{1}y\delta$.

(iii) Let g be a step-simulation of S_n on S_{n-1}. Then $T^n \circ \Pi^n(g)\ (\gamma x\delta) = \Pi^n(g)(\gamma x).\Pi^n(g)_{n-2}\ (x\delta) = g(\gamma xo)|_{n-2}.g_{n-2}(x\delta o) = g(\gamma x\delta)|_{n-2}.g_{n-1}(\gamma x\delta) = g(\gamma x\delta)$ for all γ, x, δ. Hence $T^n \circ \Pi^n = \text{id}$. Conversely, let h be a step-simulation of S_{n-1} on S_{n-2}. Then $\Pi^n \circ T^n(h)\ (\gamma x) = T^n(h)\ (\gamma xo)|_{n-2} = (h(\gamma x).h_{n-2}(xo))|_{n-2} = h(\gamma x)$ for all γ, x. Hence also $\Pi^n \circ T^n = \text{id}$. It follows that Π^n and T^n are inverses to one another when considered as operators on step-simulations.

(iv) Let g be a uniform step-simulation of S_n on S_{n-1}. Suppose

$\Pi^n(g)$ is not uniform. Then there must be a $y \in V_{n-2}$ such that $|\Pi^n(g)^{-1}(y)| > 2$. Let $x^{(1)}, x^{(2)}, x^{(3)}$ be distinct elements of $\Pi^n(g)^{-1}(y)$. It follows that $g(x^{(1)}o), g(x^{(2)}o), g(x^{(3)}o) \in \{yo,y1\}$. Because g is step-simulating we have, in fact : $g(x^{(1)}o)$, $g(x^{(1)}1)$, $g(x^{(2)}o)$, $g(x^{(2)}1)$, $g(x^{(3)}o)$, $g(x^{(3)}1) \in \{yo,y1\}$ and hence $|g^{-1}(yo)| \geq 3$ or $|g^{-1}(y1)| \geq 3$. This contradicts the uniformity of g. □

Theorem 5.3.2.3. (i)-(iii) shows that there is a 1-1 onto correspondence between the step-simulations of S_n on S_{n-1} and the step-simulations of S_{n-1} on S_{n-2}, for n≥3. Theorem 5.3.2.3. (iv) does not quite show that this correspondence holds for the subclass of uniform step-simulations, but in the next theorem we will show that it is the case.

Theorem 5.3.2.4. For n≥2,

(i) there are exactly 16 possible step-simulations of S_n on S_{n-1}.

(ii) There are exactly 6 possible uniform step-simulations of S_n on S_{n-1} (see table A).

Proof.

(i) By theorem 5.3.2.4. (i)-(iii) the number of step-simulations of S_n on S_{n-1} is equal to the number of step-simulations of S_{n-1} on S_{n-2}, for n≥3 (because Π^n is bijective). By induction this number is equal to the number of step-simulations of S_2 on S_1. Clearly every mapping $\in [V_2 \rightarrow V_1]$ is step-simulating. There are exactly $2^4 = 16$ mappings in this set.

(ii) There are exactly $\binom{4}{2} = 6$ mappings $\in [V_2 \rightarrow V_1]$ that are uniform and step-simulating. By theorem 5.3.2.4.(i)-(iv) the number of uniform step-simulations of S_n on S_{n-1} (n≥3) is not larger than the number of uniform step-simulations of S_{n-1} on S_{n-2} and thus, by induction, not larger than 6. On the other hand at least 6 uniform step-simulations of S_n on S_{n-1} can be explicitly given, see table A. (The verification of the mappings is immediate from the definition.) □

The remaining problem is to determine whether any other uniform emulation of S_n on S_{n-1} exists. Our main result is the following.

Theorem 5.3.2.5. (Characterization Theorem) Every uniform emulation of

$$f_1 : f_1(b_1..b_n) = b_1..b_{n-1}$$
$$\overline{f}_1 : \overline{f}_1(b_1..b_n) = \overline{b}_1..\overline{b}_{n-1}$$

$$f_2 : f_2(b_1..b_n) = b_2..b_n$$
$$\overline{f}_2 : \overline{f}_2(b_1..b_n) = \overline{b}_2..\overline{b}_n$$

$$f_3 : f_3(b_1..b_n) = c_1..c_{n-1} \text{ with } c_i = (b_i \equiv b_{i+1}),\ 1 \le i \le n-1$$
$$\overline{f}_3 : \overline{f}_3(b_1..b_n) = \overline{c}_1..\overline{c}_{n-1} \text{ with } c_i = (b_i \equiv b_{i+1}),\ 1 \le i \le n-1$$

Table A. Listing of the 6 possible uniform step-simulations of the 4-pin shuffle with 2^n nodes on the 4-pin shuffle with 2^{n-1} nodes.

S_n on S_{n-1} is step-simulating, and thus equal to one of the mappings listed in table A.

The proof of theorem 5.3.2.5. is long and tedious, and is given in appendix A.

5.3.3. Uniform emulations of S_n on S_{n-k}. We will extend the notion of 'step-simulation' of S_n on S_{n-k}, in order to attempt a characterization of the uniform emulations in general. We show that the step-simulations of S_n on S_{n-k} (which are not all uniform) can again be characterized in terms of the step-simulations of S_{k+1} on S_1 (cf. theorem 5.3.2.4.). It remains an open question whether all uniform emulations of S_n on S_{n-k} are step-simulating, and thus whether a suitable analogue of theorem 5.3.2.5. holds for $k \ge 1$. We show that there are at least $2.2^{2^k}-2^{2^{k-1}}$

uniform step-simulations of S_n on S_{n-k}. We also discuss the uniform emulations of $\tilde{S}_n$ on $\tilde{S}_{n-k}$.

Definition. A mapping $g : S_n \rightarrow S_{n-k}$ is called step-simulating (or : a "step-simulation" of S_n on S_{n-k}) if and only if for all $x \in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-k-1}$ and $\alpha,\beta \in \frac{o}{1}$: if $g(\alpha x) = \beta y$ then $g(x\frac{o}{1}) = y\frac{o}{1}$.

Every step-simulation clearly is an emulation (verify lemma 5.3.1.3.) and also the following analogue of lemma 5.3.2.2. holds.

Lemma 5.3.3.1. A mapping $g : S_n \rightarrow S_{n-k}$ is step-simulating if and only if for all $x \in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-k-1}$ and $\alpha,\beta \in (\frac{o}{1})$: if $g(x\alpha) = y\beta$ then $g(ox) = \frac{o}{1}y$ and $g(1x) = \frac{o}{1}y$.

We now aim for a characterization of all step-simulations of S_n on S_{n-k}.

Definition. For $n \geq k+2$, define the operators $\Pi^{n,k} : [V_n \rightarrow V_{n-k}] \rightarrow [V_{n-1} \rightarrow V_{n-k-1}]$ and $T^{n,k} : [V_{n-1} \rightarrow V_{n-k-1}] \rightarrow [V_n \rightarrow V_{n-k}]$ as follows:

$$\Pi^{n,k}(g)\ (b_1..b_{n-1}) = g(b_1..b_{n-1}o)\,|_{n-k-1}$$
$$T^{n,k}(h)\ (b_1..b_n) = h(b_1..b_{n-1}).h_{n-k-1}(b_2..b_n)$$

Theorem 5.3.3.2. For $n \geq k+2$,

(i) if g is a step-simulation of S_n on S_{n-k}, then $\Pi^{n,k}(g)$ is a step-simulation of S_{n-1} on S_{n-k-1}.

(ii) if h is a step-simulation of S_{n-1} on S_{n-k-1}, then $T^{n,k}(h)$ is a step-simulation of S_n on S_{n-k}.

(iii) restricted to step-simulations, $\Pi^{n,k}$ and $T^{n,k}$ are inverses.

(iv),restricted to step-simulations, $\Pi^{n,k}$ preserves uniformity.

Proof.

(The proof is virtually the same as for theorem 5.3.2.3. and therefore left to the reader.) □

We conclude the following results (cf. theorem 5.3.2.4.) :

Theorem 5.3.3.3. For $n \geq k+2$,

(i) there is a bijection from the set of step-simulations of S_n on S_{n-k} to the set of step-simulations of S_{n-1} on S_{n-k-1} and (hence) to the set of step-simulations of S_{k+1} on S_1.

(ii) there is an injection from the set of uniform step-simulations of S_n on S_{n-k} to the set of uniform step-simulations of S_{n-1} on S_{n-k-1} and (hence) to the set of uniform step-simulations of S_{k+1} on S_1.

Theorem 5.3.3.3. is important, as it characterizes the step-simulations of S_n on S_{n-k}. Clearly every mapping $\in [V_{k+1} \rightarrow V_1]$ is step-simulating, and thus there are precisely $2^{2^{k+1}}$ step-simulations of S_n on S_{n-k}.

Corollary 5.3.3.4. For $n \geq 1$, S_n admits precisely 2 graph-isomorphisms.

Proof.

Every isomorphism of S_n must be step-simulating. By theorem 5.3.3.3.(i) the step-simulations of S_n on S_n are in 1-1 correspondence to the step-simulations of S_1 on S_1. There are four mappings of this kind and thus precisely four step-simulations of S_n on S_n : $g_1(b_1..b_n) = b_1..b_n$, $g_2(b_1..b_n) = \bar{b}_1..\bar{b}_n$, $g_3(b_1..b_n) = o..o$, $g_4(b_1..b_n) = 1..1$. Clearly, only g_1 and g_2 are isomorphisms. □

The 1-1 correspondence referred to in theorem 5.3.3.3.(i) can be made explicit as follows. Given a step-simulation g of S_n on S_{n-k}, the uniquely corresponding step-simulation $\tilde{g}$ of S_{k+1} on S_1 is defined by the formula $\tilde{g}(b_1..b_{k+1}) = g(b_1..b_{k+1}o..o)|_1$. Conversely, given a step-simulation h of S_{k+1} on S_1, the uniquely corresponding step-simulation $\underset{\sim}{h}$ of S_n on S_{n-k} is defined by $\underset{\sim}{h}(b_1..b_n) = h(b_1..b_{k+1}) \cdot h(b_2..b_{k+2})..h(b_{n-k}..b_n)$. While the correspondence $g \rightarrow \tilde{g}$ preserves uniformity (cf. theorem 5.3.3.2. (iv)), it does not induce a bijection from the uniform step-simulations of S_n on S_{n-k} to the uniform step-simulations of S_{k+1} to S_1 for $k>1$. The existence of such a bijection for $k = 1$ (cf. theorem 5.3.2.4.(ii)) was the key to the complete characterization of the uniform step-simulations of S_n on S_{n-1} and of the uniform emulations of S_n on S_{n-1} (cf. theorem 5.3.2.5.). A similar characterization of the uniform step-simulations and of the uniform emulations of S_n on S_{n-k} for $k > 1$ remains a challenging open problem. We can characterize a large class of uniform step-simulations.

Theorem 5.3.2.5. Let $n \geq k+1$, and let g be a step-simulation of S_n on S_{n-k}.

(i) if $\tilde{g}(b_1..b_{k+1}) = \overline{\tilde{g}(\bar{b}_1 b_2..b_{k+1})}$ for all $b_1..b_{k+1} \in \binom{o}{1}^{k+1}$, then g is uniform.

(ii) if $\tilde{g}(b_1..b_{k+1}) = \overline{\tilde{g}(b_1..b_k \bar{b}_{k+1})}$ for all $b_1..b_{k+1} \in \binom{o}{1}^{k+1}$, then g is uniform.

Proof.

We only prove (i) as the proof of (ii) is similar. Induct on n. For $n = k+1$, observe from the assumption that of every pair $b_1..b_{k+1}$, $\bar{b}_1 b_2..b_{k+1}$ $\tilde{g}$ will map one to $o \in V_1$ and one to $1 \in V_1$. Thus $g = \tilde{g}$ is uniform. Assume it holds up to $n-1 \geq k+1$. Let g be a step-simulation of S_n on S_{n-k} for which the constraint on $\tilde{g}$ is satisfied. Let g' be the uniquely corresponding step-simulation of S_{n-1} on S_{n-k-1} (cf. theorem 5.3.3.3.(i)) defined by the formula $g'(b_1..b_{n-1}) = g(b_1..b_{n-1}o)|_{n-1}$. Observe that for all $b_o..b_{n-1} \in \binom{o}{1}^n$: $g(b_o b_1..b_{n-1}) = \tilde{g}(b_o b_1..b_k).\tilde{g}(b_1..b_{k+1})..\tilde{g}(b_{n-k-2}..b_{n-1})$ and likewise for $g'(b_1..b_{n-1})$, hence $g(b_o b_1..b_{n-1}) = \tilde{g}(b_o b_1..b_k).g'(b_1..b_{n-1})$. Since $\tilde{g}' = \tilde{g}$, it follows by induction that g' is uniform. Thus for every $c_1..c_{n-k-1} \in \binom{o}{1}^{n-k-1}$: $|(g')^{-1}(c_1..c_{n-k-1})| = 2^k$. Let $b_1..b_{n-1} \in (g')^{-1}(c_1..c_{n-k-1})$. By assumption it follows that of the pair $ob_1..b_k$, $1b_1..b_k$ $\tilde{g}$ will map one to $o \in V_1$ and one to $1 \in V_1$, and thus g will map one of the strings $ob_1..b_{n-1}$, $1b_1..b_{n-1}$ to $oc_1..c_{n-k-1}$ and the other to $1c_1..c_{n-k-1}$. It follows that for all $c_o c_1..c_{n-k-1} \in \binom{o}{1}^{n-k}$: $|g^{-1}(c_o c_1..c_{n-k-1})| = |(g')^{-1}(c_1..c_{n-k-1})| = 2^k$, which implies that g is uniform. This completes the inductive argument. □

Theorem 5.3.3.6. For $n \geq k+1$, there are at least $2 \cdot 2^{2^k} - 2^{2^{k-1}}$ uniform step-simulations of S_n on S_{n-k}.

Proof.

For k=1 the result follows from theorem 5.3.2.4.(ii). For k>1 we use the characterization from theorem 5.3.3.5. By induction on k one easily derives that there exist 2^{2^k} functions $\tilde{g} : V_{k+1} \to V_1$ that satisfy the constraint $\tilde{g}(b_1..b_{k+1}) = \tilde{g}(\bar{b}_1 b_2..b_{k+1})$, 2^{2^k} functions $\tilde{g} : V_{k+1} \to V_1$ that satisfy the constraint $\tilde{g}(b_1..b_{k+1}) = \tilde{g}(b_1..b_k \bar{b}_{k+1})$, and $2^{2^{k-1}}$ functions $\tilde{g}$ that satisfy both constraints simultaneously. Using the unique correspondence of $\tilde{g}$ and g, the given bound follows. □

By lemma 5.2.8. every uniform emulation $f : S_n \to S_{n-k}$ $(n,k \geq 1)$ also is a uniform emulation of $\tilde{S}_n$ on $\tilde{S}_{n-k}$, and conversely. (Note that $\tilde{S}_n = (S_n)^R$.) It follows that all results concerning the uniform emulations of S_n on S_{n-k} hold ipso facto for the uniform emulations of $\tilde{S}_n$ on $\tilde{S}_{n-k}$.

5.4. Emulations of the shuffle-exchange network. In this section we consider uniform emulations of the shuffle-exchange network (cf. section 1.6.3.). The main results, that are presented in this section are the following: we give a complete characterization of the uniform emulations of SE_n on S_{n-1} and of the graph isomorphisms of SE_n, and show there exist uniform emulations of SE_n on SE_k if k|n or k≤2, and no uniform emulations of SE_n on SE_{n-1} for n≥4.

In section 5.4.1. we give some preliminary results. In section 5.4.2. we examine the uniform emulations of SE_n on S_{n-1}. In section 5.4.3. we examine the uniform emulations of SE_n on SE_k, for k≤n. In section 5.4.4. we briefly discuss the results obtained.

5.4.1. Preliminaries.

Lemma 5.4.1.1. SE_n is a spanning subgraph of $S_{n-1}[\bar{K}_2]$ for n≥1.

Proof.

Consider the mapping $h : SE_n \to S_{n-1}[\bar{K}_2]$ defined by $h(b_1..b_n) = h(b_1..b_n) = \langle b_1..b_{n-1}, b_n \rangle$. h is clearly 1-1 and onto the set of nodes. One easily shows that h is an embedding of SE_n respectively. □

The following fact follows directly from the definitions.

Lemma 5.4.1.2. (a) f is an emulation of SE_n on SE_{n-1} if and only if for all $b \in \left(\frac{o}{1}\right)^n$, $y \in \left(\frac{o}{1}\right)^{n-k}$: if $f(b) = y$ then $\{f(b_2 \ldots b_n b_1), f(b_1 \ldots b_{n-1}\bar{b}_n)\} \subseteq \{y, y_1 \ldots y_{n-1}\bar{y}_n, y_2 \ldots y_n y_1\}$.

(b) f is an emulation of SE_n on SE_{n-1} if and only if for all $b \in \left(\frac{o}{1}\right)^n$, $y \in \left(\frac{o}{1}\right)^{n-k}$: if $f(b) = y$ then $\{ f(b_n b_1 \ldots b_{n-1}), f(b_1 \ldots b_{n-1}\bar{b}_n)\} \subseteq \{y, y_1 \ldots y_{n-1}\bar{y}_n, y_n y_1 \ldots y_{n-1}\}$. □

For a mapping f, define its companion $\bar{f}$ by $\bar{f}_i(b) = \overline{f_i(b)}$.

Lemma 5.4.1.3. If f is an emulation of S_n on S_k (or of S_n on SE_k, SE_n on S_k, SE_n on SE_k), then so if $\bar{f}$.

5.4.2. Uniform emulations of SE_n on S_{n-1}. In this section we examine the uniform emulations of SE_n on S_{n-1}. (Compare lemma 5.4.1.1.).

Lemma 5.4.2.1. Let f be a uniform emulation of SE_n on S_{n-1} with for all $x \in \left(\frac{o}{1}\right)^{n-1}$: $f(xo) = f(x1)$. Then

(a) for all $b \in \left(\frac{o}{1}\right)^n$ $f(b_1 \ldots b_n) = b_1 \ldots b_{n-1}$, or

(b) for all $b \in \left(\frac{o}{1}\right)^n$ $f(b_1 \ldots b_n) = \bar{b}_1 \ldots \bar{b}_{n-1}$.

Proof.

Define $g : S_{n-1} \rightarrow S_{n-1}$ by $g(x) = f(xo)$. g is uniform, (i.e. 1-1), because if there exist $x_1 \neq x_2$ with $g(x_1) = g(x_2)$, then $f(x_1 o) = f(x_1 1) = f(x_2 o) = f(x_2 1)$ and f is not uniform. Also g is an emulation of S_{n-1} on S_{n-1}. Suppose $g(x) = y$. Then $f(xo) = y$ and $f(x1) = y$, hence $f(ox) = y$ or $f(ox) = \frac{o}{1}y_1 \ldots y_{n-2}$, and $f(1x) = y$ or $f(1x) = \frac{o}{1}y_1 \ldots y_{n-2}$, so $g(ox_1 \ldots x_{n-2}) = f(ox_1 \ldots x_{n-2}o) \in \{y, \frac{o}{1}y_1 \ldots y_{n-2}\}$ and $g(1x_1 \ldots x_{n-2}) = f(1x_1 \ldots x_{n-2}o) \in \{y, \frac{o}{1}y_1 \ldots y_{n-2}\}$. A uniform emulation of S_{n-1} on S_{n-1} necessarily is a graph isomorphism. We now use proposition 5.3.3.4. : either for all $x \in \left(\frac{o}{1}\right)^{n-1}$ $g(x) = x$ or for all $x \in \left(\frac{o}{1}\right)^{n-1}$ $g(x) = \bar{x}$. If the former is the case, then for all $b \in \left(\frac{o}{1}\right)^n$ $f(b) = b_1 \ldots b_{n-1}$, and if the latter is the case, then for all $b \in \left(\frac{o}{1}\right)^n$ $f(b) = \bar{b}_1 \ldots \overline{b_{n-1}}$. □

Lemma 5.4.2.2. Let f be an emulation of SE_n on S_{n-1}. Then for all x ∈ $(\frac{o}{1})^{n-1}$: f(xo) = f(x1) or { f(xo),f(x1) } = { (o1)*[o],(1o)*[1] }.

Proof.

If f(xo) ≠ f(x1) then f(xo) and f(x1) must be adjacent nodes, connected by edges in both directions. The only way to realize this in S_{n-1} is to map the nodes f(xo),f(x1) onto the nodes of the set { (o1)*[o],(1o)*[1] }. □

Lemma 5.4.2.3. Let f be a uniform emulation of SE_n on S_{n-1}, and let n be odd. Then for all x ∈ $(\frac{o}{1})^{n-1}$: f(xo) = f(x1).

Proof.

Suppose the lemma does not hold. Then, by lemma 5.4.2.2., there is a x ∈ $(\frac{o}{1})^{n-1}$ such that { f(xo),f(x1) } = { (o1)*,(1o)* }.

Now suppose f((o1)*o) ∉ { (o1)*,(1o)* }. Then by lemma 5.4.2.2. f((o1)*1) = f((o1)*o). If f((1o)*o) = f((1o)*1) then f((o1)*o) can reach itself by performing two shuffle-exchange steps. (Note that (f((o1)*o),f((1o)*o)) ∈ S_{n-1} and (f((1o)*1),f((o1)*1)) ∈ S_{n-1}, due to the uniformity of f, and f((1o)*o) = f((1o)*1) and f((o1)*1)=f((o1)*o))), which forces f((o1)*o) ∈ {o^{n-1},1^{n-1}, (o1)*,(1o)* }. If f((o1)*o) = α^{n-1} = f((o1)*1), then f(o(o1)*) = f(1(o1)*) = $\bar{\alpha}\alpha^{n-2}$ (use uniformity of f). If n ≠ 3, then, by lemma 5.4.2.2. f(o(o1)*) = f(1(o1)*) = f(o(o1)*oo) = f(1(o1)*oo), which contradicts the uniformity of f. Is n = 3 then one obtains a contradiction by successively deriving that f(oo1) = f(1o1) = $\bar{\alpha}\alpha$, so f(ooo) = f(1oo) = $\alpha\bar{\alpha}$, and f(11o) = $\alpha\bar{\alpha}$ (use uniformity of f). So f((o1)*o) ∈ {(o1)*,(1o)*}. Contradiction. If f((1o)*o) ≠ f((1o)*1), then, because of lemma 5.4.2.2., f((1o)*1) ∈ { (1o)*,(o1)* }. Without loss of generality one may suppose f((1o)*1) = (1o)*, f((1o)*o) = (o1)*. Then f((o1)*o) = o(o1)*, f((o1)*1) = o(o1)* and f(1(o1)*) = $\frac{o}{1}$o(o1)*o. Contradiction. So f((o1)*o) ∈ { (o1)*,(1o)* }.

In the same way one can prove f((1o)*1) ∈ { (o1)*,(1o)* }. We have now : f^{-1}({(o1)*,(1o)*}) = { (o1)*o,(o1)*1,(1o)*o,(1o)*1 }. From the assumption there is a x ∈ $(\frac{o}{1})^{n-1}$ with { f(xo),f(x1) } = { (o1)*,(1o)* } now follows that one of the following cases must hold : I.f((o1)*o) =

$(o1)^* \wedge f((o1)^*1) = (1o)^*$; II. $f((o1)^*o) = (1o)^* \wedge f((o1)^*1) = (o1)^*$; III. $f((1o)^*1) = (o1)^* \wedge f((1o)^*o) = (1o)^*$; IV $f((1o)^*1) = (1o)^* \wedge f((1o)^*o) = (o1)^*$. We will only handle case I; the other cases are similar.

So suppose $f((o1)^*o) = (o1)^*$ and $f((o1)^*1) = (1o)^*$. With downward induction on k we prove : for all $x \in (\frac{o}{1})^{n-2k-2}$ there is a $y \in (\frac{o}{1})^{n-2k-3}$ such that $f((o1)^k oox) = (o1)^k ooy$. First we prove this fact for $n-2k-2=1$ i.e. $k=\frac{1}{2}(n-3)$. $f((o1)^*o) = (o1)^* \Rightarrow f(o(o1)^*) = o(o1)^*o \Rightarrow f(o(o1)^*oo) = o(o1)^*o \Rightarrow f((o1)^*oo\frac{o}{1}) = (o1)^*oo$ (use lemma 5.4.2.2. and the uniformity of f). Now let the proposition be true for a certain k. $f((o1)^k oox) = (o1)^k ooy$, so $f(1(o1)^{k-1}oox\frac{o}{1}) = (1o)^{k-1}1ooy\frac{o}{1}$. (Notice that $f((o1)^k oox) = f((o1)^k oox_1 \ldots x_{n-2k-3}\overline{x}_{n-2k-2})$, and due to the uniformity of f one gets $f(1(o1)^{k-1}oox\frac{o}{1}) \neq f((o1)^k oox)$. Using basically the same argument one proves $f((o1)^{k-1}oox\frac{oo}{11}) = (o1)^{k-1}ooy\frac{oo}{11}$, thus completing the inductional proof of the proposition.

In particular we now have $f(\ \{oox \mid x \in (\frac{o}{1})^{n-2}\}\ ) = \{\ ooy \mid y \in (\frac{o}{1})^{n-3}\ \}$. Now $f((o1)^*1) = (1o)^* \Rightarrow f((1o)^*11o) = (o1)^*oo$. With downward induction on k we prove $f((1o)^k 11o(1o)^{(n-2k-3)/2}) = (o1)^k ooy$, for some $y \in (\frac{o}{1})^{n-2k-3}$. We already proved this to be true for $k=(n-2k-3)/2$. Now let it be true for certain k. Notice $f((1o)^k 11o(1o)^*) = f((1o)^k 11o(1o)^*11)$, so $f(o(1o)^{k-1}11o(1o)^*1) \neq f((1o)^k 11o(1o)^*)$, so $f(o(1o)^*11o(1o)^*1) = 1(o1)^{k-1}ooy'$ for some $y' \in (\frac{o}{1})^{n-2k-2}$. Using the same type of argument one proves $f((1o)^{k-1}11o(1o)^*) = (o1)^{k-1}ooy''$ for some $y'' \in (\frac{o}{1})^{n-2k-1}$.

This shows that $f^{-1}(\ \{ooy \mid y \in (\frac{o}{1})^{n-3}\}\ ) \supseteq \{oox \mid x \in (\frac{o}{1})^{n-2}\} \cup \{11o(1o)^*\}$, which contradicts the uniformity of f. □

Lemma 5.4.2.4. Let f be a uniform emulation of SE_n on S_{n-1}. Let n be even. Let $\tilde{f} : SE_n \to S_{n-1}$ be defined by $\tilde{f}((o1)^*) = f((1o)^*)$, $\tilde{f}((1o)^*) = f((o1)^*)$ and $\tilde{f}(b) = f(b)$, if $b \notin \{\ (o1)^*, (1o)^*\ \}$. Then either for all $x \in (\frac{o}{1})^{n-1}$: $f(xo) = f(x1)$ or for all $x \in (\frac{o}{1})^{n-1}$: $\tilde{f}(xo) = \tilde{f}(x1)$. In the latter case $\tilde{f}$ is a uniform emulation function of SE_n on S_{n-1}.

Proof.

Note that f((o1)*) and f((1o)*) must be adjacent with edges in both directions, or equal. If f((o1)*) = f((1o)*), then f((o1)*oo) ≠ f((o1)*) by uniformity, hence { f((o1)*oo), f((o1)*) } = { (o1)*o, (1o)*1 }, and f((1o)*11) ≠ f((1o)*), hence { f((1o)*11), f((1o)*) } = {(o1)*o, (1o)*1}. If f((o1)*) ≠ f((1o)*), then f((o1)*) and f((1o)*) must be mutually adjacent to each other, so { f((o1)*), f((1o)*) } = { (o1)*o, (1o)*1 }. In both cases one has f^{-1} {(o1)*o, (1o)*1} = {(o1)*, (1o)*, (o1)*oo, (1o)*11 }.

Now suppose there is a x ∈ $(\frac{o}{1})^{n-1}$ such that f(xo) ≠ f(x1). Then one of the following four cases must hold : I. f((o1)*) = (o1)*, f((o1)*oo) = (1o)*1, II. f((o1)*) = (1o)*1, f((o1)*oo) = (o1)*o, III. f((1o)*) = (o1)*o, f((1o)*11) = (1o)*1, IV. f((1o)*) = (1o)*1, f((1o)11) = (o1)*o. We will only examine case I; the other cases are similar.

So suppose f((o1)*) = (o1)*o, and f((o1)*oo) = (1o)*1. Note that, due to the uniformity of f, f(o(o1)*o) ≠ (o1)*o and f(o(o1)*o) ≠ (1o)*o, so f(o(o1)*o) = 1(1o)*. Now f(o(o1)*1) = 1(1o)*, f((o1)*1o) = (1o)*o and f((o1)*11) = (1o)*o. So f((1o)*11) = (o1)*o, and f((1o)*) = (1o)*1. Notice that (1o)* is only adjacent to (o1)* and (1o)*11, and (o1)* is only adjacent to (1o)* and (o1)*oo in SE_n; each of these nodes is mapped to the set { (o1)*o, (1o)*1 }, so $\tilde{f}$ is also an emulation function. It is easy to verify (using lemma 5.4.2.2.), that for all x ∈ $(\frac{o}{1})^{n-1}$ $\tilde{f}$(xo) = $\tilde{f}$(x1), and that $\tilde{f}$ is uniform. □

Theorem 5.4.2.5. (Characterization Theorem).

(a) If n is odd, then every uniform emulation of SE_n on S_{n-1} is one of the following list:

$f \quad : \quad f(b_1 \ldots b_n) \;=\; b_1 \ldots b_{n-1}$

$\overline{f} \quad : \quad \overline{f}(b_1 \ldots b_n) \;=\; \overline{b}_1 \ldots \overline{b}_{n-1}$

(b) If n is even, then every uniform emulation of SE_n on S_{n-1} is one of the following list :

$f_1 \quad : \quad f_1(b_1 \ldots b_n) = b_1 \ldots b_{n-1}$

$\overline{f}_1 \quad : \quad \overline{f}_1(b_1 \ldots b_n) = \overline{b}_1 \ldots \overline{b}_{n-1}$

f_2 : $f_2(b_1 \ldots b_n) = b_1 \ldots b_{n-1}$ if $b \in \{ (o1)^*, (1o)^* \}$
$f_2((o1)^*) = (1o)^*1$
$f_2((o1)^*) = (o1)^*o$

$\bar{f}_2$: $\bar{f}_2(b_1 \ldots b_n) = \bar{b}_1 \ldots \bar{b}_{n-1}$ if $b \notin \{ (o1)^*, (o1)^* \}$
$\bar{f}_2((o1)^*) = (o1)^*o$
$\bar{f}_2((1o)^*) = (1o)^*1$.

Proof.

(a) Use lemma 5.4.2.1. and 5.4.2.3.

(b) Use lemma 5.4.2.4. If for all $x \in (\frac{o}{1})^{n-1}$ $f(xo) = f(x1)$, then f is of the form f_1 or $\bar{f}_1$ (use lemma 5.4.2.1). If for all $x \in (\frac{o}{1})^{n-1}$ $\tilde{f}(xo) = \tilde{f}(x1)$, then $\tilde{f}$ is of the form f_1 or $\bar{f}_1$, so f is of the form f_2 or $\bar{f}_2$. □

5.4.3. Uniform emulations of SE_n on SE_k $(k \leq n)$.

Proposition 5.4.3.1. For $n \geq 1$, SE_n admits precisely 2 graph isomorphisms.

Proof.

If $n=1,2$, verify directly.

Let $n \geq 3$. Clearly g, defined by $g(b) = b$ and $\bar{g}$, defined by $\bar{g}(b) = \bar{b}$ are isomorphisms. Suppose there is yet another isomorphism of SE_n, $\tilde{g}$. Define $h(b) = \tilde{g}(b)|_{n-1}$. h is a uniform emulation of SE_n on S_{n-1}. We use theorem 5.4.2.5. and consider 4 cases :

Case I : for all $b \in (\frac{o}{1})^n$: $h(b) = b_1 \ldots b_n$.

Because $\tilde{g} \neq g$, there must be a b with $\tilde{g}(b) = b_1 \ldots b_{n-1}\bar{b}_n$. Now $\tilde{g}(b_n b_1 \ldots b_n) = b_n b_1 \ldots b_{n-2} \frac{o}{1}$ must be adjacent to $b_1 \ldots b_{n-1}\bar{b}_n$ in SE_n. So $b_1 \ldots b_n = \alpha^n$, for some $\alpha \in (\frac{o}{1})$, and $g(\alpha^n) = \alpha^{n-1}\bar{\alpha}$. Now note that the outdegree of α^n is 1 and the outdegree of $\alpha^{n-1}\bar{\alpha}$ is 2 in SE_n, so $\tilde{g}$ is not a graph isomorphism. Contradiction.

Case II : for all $b \in (\frac{o}{1})^n$: $h(b) = \bar{b}_1 \ldots \bar{b}_{n-1}$.

This case can be handled in the same way as case 1.

Case III : n is even, $h(b) = b_1 \ldots b_{n-1}$ if $b \notin \{(o1)^*, (o1)^*\}$, $h((o1)^*) = (1o)^*1$, $h((1o)^*) = (o1)^*o$.

If there is a b with $\tilde{g}(b) = b_1 \ldots b_{n-1}\bar{b}_n$, then we reach a contradiction in the same way as in case 1. So we may suppose $\tilde{g}(b) = b$ for all $b \notin \{ (o1)^*, (1o)^* \}$. So $\tilde{g}((o1)^*) = (1o)^*$, $\tilde{g}((1o)^*) = (o1)^*$, and now $\tilde{g}((o1)^*)$ is not adjacent to $\tilde{g}((o1)^*oo)$. Contradiction.

Case IV : n is even, $h(b) = \bar{b}_1 \ldots \bar{b}_{n-1}$, if $b \notin \{ (o1)^*, (1o)^* \}$, $h((o1)^*) = (o1)^*o$, $h((1o)^*) = (1o)^*1$.

This case can be handled in the same way as case 3. It follows that there are no other graph isomorphism of SE_n but g and $\bar{g}$. □

Theorem 5.4.3.2. Let $k,n \geq 1$, $k|n$. Then the function f, defined by $f_i(b_1 \ldots b_n) = (\sum_{j=o}^{n/k-1} b_{j.k+i}) \bmod 2$ $(i=1,\ldots,k)$ is a uniform emulation of SE_n on SE_k.

Proof.

By verifying that $f_i(b_1 \ldots b_n) = f_i(b_1 \ldots b_{n-1}\bar{b}_n)$ for $i \neq k$ and that $f_i(b_1 \ldots b_n) = f_{i+1}(b_2 \ldots b_n b_1)$ for $i \neq k$ and $f_k(b_1 \ldots b_n) = f_1(b_2 \ldots b_n b_1)$ one proves that f is an emulation of SE^n on SE^k. If x and $y \in SE_k$ differ only in the i'th bit position, then $f^{-1}(y) = (b_1 \ldots b_{i-1}\bar{b}_i b_{i+1} \ldots b_n \mid b \in f^{-1}(x) \}$, so $| f^{-1}(x) | = |f^{-1}(y)|$. With induction one now can prove that for all $x,y \in (\frac{o}{1})^k$ $|f^{-1}(x)| = |f^{-1}(y)|$, so f is uniform. □

Proposition 5.4.3.3. Let $n \geq 2$. Then SE_n can be uniformly emulated on SE_2.

Proof.

If n is even, then use theorem 5.4.3.2.

Let n be odd. The graph SE_2 is shown in fig. 5.4.3.1.

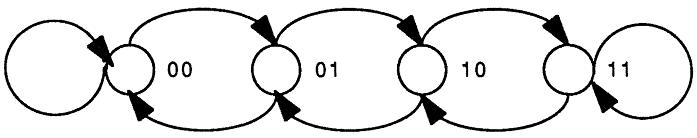


fig. 5.4.3.1. SE_2

Let f be a mapping of SE_n on SE_2. We will show that f can be chosen such that f is a uniform emulation. We first want f to fulfill the following conditions :

$$f_1(b_1...b_n) = \begin{cases} o \text{ if } \sum_{i=1}^{n} b_i < \frac{1}{2}n \\ \\ 1 \text{ if } \sum_{i=1}^{n} b_i > \frac{1}{2}n \end{cases}$$

f maps one half of the nodes of SE_n on {oo,o1} and the other half on {10,11}. We can choose f in such a way that every string with $\sum_{i=1}^{n} b_i = \lfloor\frac{1}{2}n\rfloor$ and b_n = o is mapped to o1 and every string with $\sum_{i=1}^{n} b_i = \lceil\frac{1}{2}n\rceil$ and b_n = 1 is mapped to lo and f is uniform. We show that f is an emulation of SE_n on SE_2: If b,b′ are adjacent in SE_n, then there are four cases : I. $\sum_{i=1}^{n} b_i \le \lfloor\frac{1}{2}n\rfloor$ and $\sum_{i=1}^{n} b_i' \le \lfloor\frac{1}{2}n\rfloor$, then b, b′ are mapped to nodes in the set {oo,o1}; II. $\sum_{i=1}^{n} b_i \le \lfloor\frac{1}{2}n\rfloor$ and $\sum_{i=1}^{n} b_i' \ge \lceil\frac{1}{2}n\rceil$, then necessarily b=xo and b′=x1 for some $x \in (\frac{o}{1})^{n-1}$ and $\sum_{i=1}^{n-1} x_i = \lfloor\frac{1}{2}n\rfloor$, so b is mapped upon o1 and b′ is mapped upon lo; III. $\sum_{i=1}^{n} b_i \ge \lceil\frac{1}{2}n\rceil$, and $\sum_{i=1}^{n} b_i' \le \lfloor\frac{1}{2}n\rfloor$; this case is similar to case II; IV. $\sum_{i=1}^{n} b_i \ge \lceil\frac{1}{2}n\rceil$ and $\sum_{i=1}^{n} b_i' \ge \lceil\frac{1}{2}n\rceil$, then $\{f(b_i),f(b_i')\} \subseteq \{lo,11\}$. So f is a uniform emulation of SE_n on SE_2. □

For choices of n and k other than k|n and k≤2 there are presently no uniform emulation functions of SE_n on SE_k known. We conjecture that for n,k>2 with n>k, $k \nmid n$, no such function exists. The following results show that the conjecture is at least plausible. We show that from a uniform emulation of SE_n on SE_k with n>k>2 and $k \nmid n$, an emulation function of S_{n-1} on S_{k-1} can be derived, that is uniform, but not step-simulating. Presently no functions of this sort are known. We also show that for n≥4, there indeed are no uniform emulations of SE_n on SE_{n-1}.

<u>Lemma 5.4.3.4</u>. Let f be an emulation function $SE_n \rightarrow SE_k$ (n,k≥1).

(a) If k is odd, then for all $x \in (\frac{o}{1})^{n-1}$: $f(xo)|_{k-1} = f(x1)|_{k-1}$.

(b) If k is even, then for all $x \in (\frac{o}{1})^{n-1}$: $f(xo)|_{k-1} = f(x1)|_{k-1}$ or $\{f(xo), f(x1)\} = \{(o1)^*, (1o)^*\}$.

Proof.

f(xo) and f(x1) must be adjacent or equal. □

Lemma 5.4.3.5. Let f be an emulation of SE_n on SE_k ($n,k \geq 2$). Let ψ be a function $(\frac{o}{1})^{n-1} \rightarrow (\frac{o}{1})$. Let $g : S_{n-1} \rightarrow S_{k-1}$ be defined by $g(x) = f(x\psi(x))|_{k-1}$ (for all $x \in (\frac{o}{1})^{n-1}$). Then g emulates S_{n-1} on S_{k-1}.

Proof.

We first show that $g(\psi(x)(x|_{n-2}))$ is adjacent to g(x). Let $g(x) = y$. Then $f(x\psi(x)) = y\frac{o}{1}$, so $f(\psi(x)x) = y\frac{o}{1}$ or $f(\psi(x)x) = \frac{o}{1}y$. If $x_{n-1} = \psi(\psi(x)(x|_{n-2}))$, then $g(\psi(x)(x|_{n-2})) = y$ or $g(\psi(x)(x|_{n-2})) = \frac{o}{1}(y|_{k-1})$, and adjacency is proved. If $x_{n-1} = \overline{\psi(\psi(x)(x|_{n-2}))}$ then either $f(\psi(x)(x|_{n-2})o)|_{n-1} = f(\psi(x)(x|_{n-2})1)|_{n-1}$ or $\{ f(\psi(x)(x|_{n-2})o), f(\psi(x)(x|_{n-2})1)\} = \{(o1)^*, (1o)^*\}$. (In the latter case k must be even.) In the former case $g(\psi(x)(x|_{n-2})) \in \{y, \frac{o}{1}(y|_{k-1})\}$, and adjacency follows. In the latter case $g(\psi(x)(x|_{n-2})) \in \{(o1)^*o, (1o)^*1\}$, and $f(\psi(x)x) \in \{(o1)^*, (1o)^*\}$, hence $f(x\psi(x)) \in \{(o1)^*, (1o)^*, (o1)^*oo, (1o)^*11\}$ and $g(x) \in \{(o1)^*o, (1o)^*1\}$, and adjacency follows again.

Next we verify that $g(\overline{\psi(x)}(x|_{n-2}))$ is adjacent (or equal) to g(x). Let $g(x) = y$. Then $f(x\psi(x)) = y\frac{o}{1}$. If $f(x\psi(x))|_{k-1} = f(x\overline{\psi(x)})|_{k-1}$, then $f(\overline{\psi(x)}x) = y\frac{o}{1}$ or $f(\overline{\psi(x)}x) = \frac{o}{1}y$ and the argument can proceed as in the first part of the proof. So suppose $\{f(x\psi(x)), f(x\overline{\psi(x)}\} = \{(o1)^*, (1o)^*\}$. Now $g(x) \in \{(o1)^*o, (1o)^*1\}$ and $f(\overline{\psi(x)}x) \in \{ (o1)^*, (1o)^*, (o1)^*oo, (1o)^*11\}$. Hence $f(\overline{\psi(x)}(x|_{n-2})\ \psi(\overline{\psi(x)}(x|_{n-2}))) \in \{(o1)^*, (1o)^*, (o1)^*oo, (1o)^*11\}$ and $g(\overline{\psi(x)}(x|_{n-2})) \in \{(o1)^*o, (1o)^*1\}$. Adjacency now follows again. □

Lemma 5.4.3.6. Let f be a uniform emulation of SE_n on SE_k ($n>k$). There exists a $\psi : (\frac{o}{1})^{n-1} \rightarrow (\frac{o}{1})$, such that the function g, defined by $g(x) = f(x\psi(x))|_{k-1}$ is a uniform emulation of S_{n-1} on S_{k-1}.

Proof.

Lemma 5.4.3.5 shows that g is an emulation for every choice of ψ. So we have to show that ψ can be chosen such that g is uniform. If k is odd, then any $\psi : (\frac{o}{1})^{n-1} \to (\frac{o}{1})$ will do. Suppose g is not uniform. Then there is an $x \in (\frac{o}{1})^{k-1}$ with $|g^{-1}(x)| \neq 2^{n-k}$. One has $g(b) = x \Leftrightarrow f(b\psi(b)) \in \{xo,x1\} \Leftrightarrow f(b\overline{\psi(b)}) \in \{xo,x1\}$ (use lemma 5.4.3.4.), so $|f^{-1}(\{xo,x1\})| = 2.|g^{-1}(x)| \neq 2^{n-k+1}$. So f is not uniform. Contradiction.

Now suppose k is even. For every $x \in (\frac{o}{1})^{n-1}$ with $f(xo)|_{k-1} = f(x1)|_{k-1}$ we can choose $\psi(x)$ arbitrarily. Let $X = \{x \in (\frac{o}{1})^{n-1} | f(xo)|_{k-1} \neq f(x1)|_{k-1}\} = \{x \in (\frac{o}{1})^{n-1} | \{f(xo),f(x1)\} = \{(o1)^*,(1o)^*\}\}$. There must be an even number of x with $\{f(xo),f(x1)\} = \{(o1)^*,(o1)^*oo\}$, else there would be an odd number of nodes mapped upon $(o1)^*oo$. Likewise there must be an even number of x with $\{f(xo), f(x1)\} = \{(1o)^*,(1o)^*11\}$. Hence there must be an even number of x with $\{f(xo),f(x1)\} = \{(o1)^*,(1o)^*\}$. $|X|$ is even. Choose $X_1,X_2 \subseteq X$, such that $|X_1|=|X_2|$, $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$.) For $x \in X_1$ we choose $\psi(x)$, such that $f(x\psi(x)) = (o1)^*$. (This is possible, because $\{f(xo),f(x1)\} = \{(o1)^*,(1o)^*\}$. For $x \in X_2$ we choose $\psi(x)$, such that $f(x\psi(x)) = (1o)^*$. Now g is uniform. For $y \notin \{(o1)^*o, (1o)^*1\}$, $g(b) = y \Leftrightarrow \{f(bo),f(b1)\} \subseteq \{yo,y1\}$, so $2^{n-k+1} = |f^{-1}(\{yo,y1\})| = 2|g^{-1}(y)|$, and $|g^{-1}(y)| = 2^{n-k}$. If $g(b) = (o1)^*o$, then either $b \in \{x| \{f(xo),f(x1)\} \subseteq \{(o1)^*, (o1)^*oo\}\} = Z_1$ or $b \in X_1$. If $g(b) = (1o)^*1$, then either $b \in \{x|\{f(xo), f(x1)\} \subseteq \{(1o)^*, (1o)^*11\}\} = Z_2$ or $b \in X_2$. Finally notice that $|Z_1| = |Z_2|$ and $|X_1|=|X_2|$. Hence $|g^{-1}(\{(o1)^*o\})| = |g^{-1}(\{(10)^*1\})|$, which shows that $|g^{-1}(y)| = 2^{n-k}$, for all $y \in (\frac{o}{1})^{k-1}$. □

Lemma 5.4.3.7. Let f be an emulation of SE_n on SE_k and let f be surjective*. Let $n \geq k \geq 3$ and $k \not| n$. Let ψ be a function $(\frac{o}{1})^{n-1} \to (\frac{o}{1})$, and let g be the emulation of S_{n-1} on S_{k-1} defined by $g(x) = f(x\psi(x))|_{k-1}$. Then g is not step-simulating.

* Note that every uniform emulation function is surjective.

Proof.

Suppose g is step-simulating. We use the notation $R^{l}(b)$ to denote the string obtained by rotating b l bits to the left, i.e. $R^{l}(b_1...b_n) = b_{l+1}...b_n b_1..b_l$. Let $f(b) = o^{k-2}11$ for certain $b \in (\frac{o}{1})^n$.

With induction we prove : for all l $o\leq l\leq n$ $f(R^{l}(b)) = R^{l}(f(b))$. For l = o this is trivially true. Suppose $f(R^{l}(b)) = R^{l}(f(b))$ for certain l. Then $f(R^{l}(b))|_{n-1}\psi(R^{l}(b))) = R^{l}(f(b))|_{k-1}\frac{o}{1}$ (notice that $R^{l}(f(b)) \in \{(o1)^*,(1o)^*\})$, hence $g(R^{l}(b)|_{n-1}) = R^{l}(f(b))|_{k-1}$. g is step-simulating, so $g(_2|R^{l}(b)) = {}_2|(R^{l}(f(b))|_{k-1})\frac{o}{1}$ and $f(_2|R^{l}(b)(\psi(_2|R^{l}(b))))$ $= {}_2|(R^{l}(f(b))|_{k-1})\frac{o}{1}\frac{o}{1}$. Notice that for even k $f(R^{l+1}(b)) = f(_2|R^{l}(b)R_1^{\,l}(b)) \notin \{(o1)^*, (1o)^* \}$, because $f(R^{l+1}(b))$ must be adjacent to $f(R^{l}(b)) = R^{l}(o^{k-2}11)$ in SE_k. So $f(R^{l+1}(b)) = f(_2|R^{l}(b)R_1^{\,l}(b)) = {}_2|(R^{l}(f(b))|_{k-1})\frac{o}{1}\frac{o}{1}$ and, because $f(R^{l+1}(b))$ is adjacent to $f(R^{l}(b))$, $f(R^{l+1}(b)) = {}_2|(R^{l}(f(b))|_{k-1})R_k^{l}(f(b))R_1^{l}(f(b)) = R^{l+1}(f(b))$. This completes the inductive proof.

In particular we now have $f(R^{n}(b)) = R^{n}(f(b)) \Rightarrow f(b) = R^{n}(f(b)) \Rightarrow k|n$, contradiction. □

The lemma indicates that it is not very likely that for $k\nmid n$ and $n>k\geq 3$ there exist uniform emulations of SE_n on SE_k, and that if they do exist, they will probably not have a nice structure. Presently no uniform emulations of S_n on S_{n-k} are known that are not step-simulating. As a corollary we have:

Theorem 5.4.3.8. There exist no uniform emulations of SE_n on SE_{n-1}, for $n\geq 4$.

Proof.

Suppose there exists a uniform emulation of SE_n on SE_{n-1}, for some $n\geq 4$. Then, by lemmas 5.4.3.5., 5.4.3.6. and 5.4.3.7. there exists a uniform emulation of S_{n-1} on S_{n-2}, that is not step-simulating. This contradicts theorem 5.3.2.5. □

5.4.4. Discussion. The 4-pin shuffle with 2^n nodes can be emulated on the 4-pin shuffle with 2^k nodes for all $k \in \{o,..., n\}$, whereas the shuffle-exchange network with 2^n nodes cannot always be emulated on smaller shuffle-exchange networks. This indicates that from the (important) viewpoint of emulation the 4-pin shuffle is preferable over the (classical) shuffle-exchange network.

5.5. Emulations of the cube network. In this section we consider emulations of the cube network. Recall the definition of the cube network (cf. section 1.6.4.). Our main result will be a complete characterization of the uniform emulations of C_n on C_{n-1}, in terms of the uniform emulations of C_3 on C_2. This section will be devoted to various auxiliary results and the proof of the main theorem. The argument depends on a crucial lemma (theorem 5.5.5.) whose lengthy proof is deferred to appendix C.

In this section we use b,c,.. to denote full addresses and x,y.. to denote segments of bits. The i^{th} bit of an address b is denoted by b_i ($1 \le i \le n$). For $|x| = |y|$, let $d(x,y)$ be the Hamming distance between the bitstrings x and y, i.e., the number of bit-positions in which x and y differ. (See, for example, Deo [D74] sect. 12-5)

The fact that C_n can be (uniformly) emulated on every C_{n-k} for $k \ge 1$ easily derives from the following observation, using lemma 5.2.2.

Proposition 5.5.1. For $k \ge 1$, C_n is isomorphic to a spanning subgraph of $C_{n-k}[C_k]$.

Proof.

One verifies that the mapping $h : C_n \to C_{n-k}[C_k]$ defined by the formula $h(b_1..b_n) = < b_1..b_{n-k}, b_{n-k+1}..b_n >$ is a subgraph isomorphism. □

Lemma 5.5.2. f is an emulation of C_n on C_{n-k} if and only if for all b, c $\in V_n$: if $d(b,c) = 1$ then $d(f(b),f(c)) \le 1$.

(The proof follows directly from the definition of emulation. Note that the condition can be equivalently written as: $d(f(b),f(c)) \le d(b,c)$.) We shall be interested in characterizing the uniform emulations of C_n on C_{n-1}.

The distinguishing feature of C_n is that it admits many more isomorphisms than e.g. S_n (cf. corollary 5.3.3.4.). This immediately has consequences for the characterization of uniform emulations, because of the following fact.

Lemma 5.5.3. Let I, I′ be isomorphisms of C_n, C_{n-1} respectively. For every f, if f is a uniform emulation of C_n on C_{n-1} then so is $I' \circ f \circ I$ (and conversely).

(The easy proof of lemma 5.5.3. is left as an exercise.) The isomorphisms of C_n can be characterized. For permutations Π let I_Π be the isomorphism defined by $I_\pi(b_1..b_n) = b_{\pi(1)}..b_{\pi(n)}$, and for index sets J ⊆ {1,...,n} let I_J be the isomorphism defined by

$$(I_J)_i\ (b_1..b_n) = \begin{cases} \bar{b}_i & \text{if } i \in J \\ b_i & \text{otherwise} \end{cases}$$

for $1 \le i \le n$. Thus, I_J flips the bits in the positions with index in J.

Theorem 5.5.4. I is an isomorphism of C_n if and only if there are J, Π such that $I = I_J \circ I_\Pi$.

Proof.

The "if"-part is obvious. To prove the "only-if"-part, proceed as follows. Consider I(o..o) and choose J such that i ∈ J if and only if I_i(o..o) = 1. Furthermore choose Π such that if the i^{th} bit of o..o is flipped, then so is the $\Pi(i)^{th}$ bit of I(o..o). Observe that such a permutation Π must exist. Define the weight w(b) of a bitstring b as the number of nonzero bits in b. We prove by induction on w(b) that for all $b \in V_n : I_J^{-1} \circ I = I_\Pi$. For $w(b) \le 1$ it holds : observe that $I_J^{-1} \circ I(o..o) = (o..o)$ and that if the i^{th} bit of o..o is flipped, then so is the $\Pi(i)^{th}$ bit of $I_J^{-1} \circ I(o..o)$. Suppose it holds for all b with $w(b) \le m$ for some $m \ge 1$. Consider $b \in V_n$ with $w(b) = m+1$ and choose $c, c' \in V_n$ of weight m, with $c \ne c'$ and $d(b,c) = d(b,c') = 1$. Suppose b is obtained from c,c′ by flipping the i^{th}, j^{th} bit from o to 1 respectively, for some i≠ j. By induction $I_J^{-1} \circ I(c) = I_\Pi(c)$ and $I_J^{-1} \circ I(c') = I_\Pi(c')$ and clearly $I_\Pi(c)$ and $I_\Pi(c')$ differ in the $\Pi(i)^{th}$ and $\Pi(j)^{th}$ position. If $I_J^{-1} \circ I(b)$ is obtained from $I_\Pi(c)$ by flipping a bit in a position $\notin \{\Pi(i),\Pi(j)\}$ then it will have a distance ≥ 2 from $I_\Pi(c')$. Contradiction. Suppose $I_J^{-1} \circ I(b)$ is obtained from $I_\Pi(c)$ by flipping the $\Pi(j)^{th}$ bit. Clearly $c_j = 1$. Let c" be the string obtained from c by setting the j^{th} bit to o. $I_\Pi(c")$ is obtained from $I_\Pi(c)$ by flipping the $\Pi(j)^{th}$ bit, so $I_J^{-1} \circ I(b) = I_\Pi(c")$. It follows that w(c") = m−1 and (hence) b ≠ c" and (by induction) $I_J^{-1} \circ I(b) = I_\Pi(c") = I_J^{-1} \circ I(c")$, contradicting that $I_J^{-1} \circ I$ is 1-1. Thus $I_J^{-1} \circ I(b)$ is obtained from $I_{\Pi(c)}$ by flipping the $\Pi(i)^{th}$ bit and thus $I_J^{-1} \circ I(b) = I_\Pi(b)$. This completes the induction. We conclude that $I_J^{-1} \circ I = I_{\Pi'}$ or $I = I_J \circ I_{\Pi'}$. □

Viewing C_n as the n-dimensional unit cube brings the analysis of emulations into the realm of combinatorial topology.

Definition. For o≤m≤n, an m-face of C_n is any subgraph (subcube) of 2^m nodes of C_n that have identical bits in n−m corresponding positions.

Crucial for the characterization of uniform emulations is the following result, the proof of which is deferred to appendix C.

Theorem 5.5.5. (Topological Reduction Theorem). Let n≥4, and let f be a uniform emulation of C_n on C_{n-1}. Then there exists an (n−1)-face A of C_n such that f(A) is an (n−2)-face of C_{n-1}.

Definition. For mappings $g : V_3 \rightarrow V_2$, let $\hat{g} : V_n \rightarrow V_{n-1}$ be the mapping defined by $\hat{g}(b_1..b_n) = g(b_1b_2b_3)b_4..b_n$ (n≥4).

Theorem 5.5.6. (Characterization Theorem). For n≥3, f is a uniform emulation of C_n on C_{n-1} if and only if there are isomorphisms I and I′ of C_n and C_{n-1} respectively and a uniform emulation g of C_3 on C_2 such that $f = I' \circ \hat{g} \circ I$.

Proof.

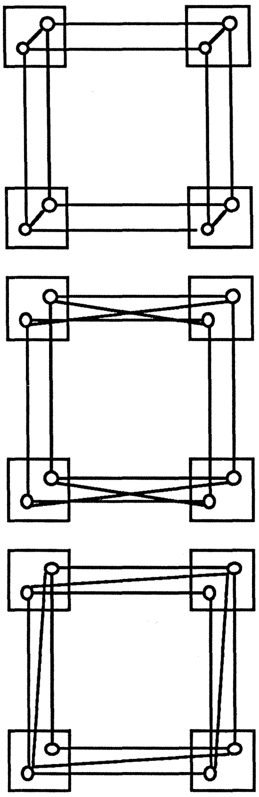
The "if"-part is obvious. For the "only if"-part we induct on n. The characterization is obvious for n=3. Assume it holds up to $n-1\geq3$, and consider a uniform emulation f of C_n on C_{n-1}. By theorem 5.5.5. there is an (n-1)-face A of C_n such that f(A) is an (n-2)-face of C_{n-1}. Up to isomorphisms of C_n and C_{n-1} we may assume that A is determined by elements b that have identical b_n and that f(A) is determined by elements c that have identical c_{n-1}. Because of uniformity no elements of the complementary face A^c (i.e., the elements with bit b_n flipped) can be mapped into f(A). It follows that A^c is mapped to $f(A)^c$ (i.e., the elements of f(A) with bit c_{n-1} flipped) and, because f emulates, that $f(b_1..b_{n-1}b_n)$ and $f(b_1..b_{n-1}\overline{b}_n)$ are equal in the first n-2 bits for all $b_1..b_n \in V_n$. It follows that, restricted to $A \cong V_{n-1}$, f reduces to a mapping f' depending on $b_1..b_{n-1}$ only and $f(b_1..b_n) = f'(b_1..b_{n-1})b_n$ for all $b_1..b_n \in V_n$ or $f(b_1..b_n) = f'(b_1..b_{n-1})\overline{b}_n$ for all $b_1..b_n \in V_n$. Up to another isomorphism of C_{n-1} we can assume the former. As a mapping from $A \cong V_{n-1}$ to $f(A) \cong V_{n-2}$, f' is seen to act as a uniform emulation of C_{n-1} on C_{n-2}. The induction hypothesis now applies to obtain the desired form for f. □

The characterization of theorem 5.5.6. is complete once the uniform emulations of C_3 on C_2 are explicitly given. Clearly there are many that are similar, by lemma 5.5.3. Characterized by the smallest m such that an m-face is mapped to an (m-1)-face, only three essentially different uniform emulations of C_3 on C_2 can arise. The different types are given in table B.

It is open whether a similar, complete characterization can be given of the uniform emulations of C_n on C_{n-k} for $k > 1$.

5.6. Emulations of the ring and the two-dimensional grid network.

Throughout this section let n be even, unless stated otherwise. Let R_n be the ring network with n nodes, and let GR_n be the n x n grid network (with n^2 nodes) with wrap-around connections (cf section 1.6.1 and 1.6.2.). In section 5.6.1. we give a complete characterization of the



Type 1 : 1-face → o-face

Type 2 : 2-face → 1-face

Type 3 : 3-face → 2-face

Table B. Classification of the uniform emulations of C_3 on C_2 according to the smallest m for which an m-face is mapped to an (m-1)-face.

uniform emulations of R_n on $R_{n/2}$. In section 5.6.2. we show that the number of uniform emulations of GR_n on $GR_{n/2}$ is at least exponential in n.

5.6.1. Uniform emulations of R_n on $R_{n/2}$. Recall the definition of the ring network R_n from section 1.6.1. By "wrapping" it around $R_{n/2}$ twice, it follows that R_n can be uniformly emulated on $R_{n/2}$. Our aim will be to characterize all possible uniform emulations of R_n on $R_{n/2}$.

It will be helpful to view R_n (hence $R_{n/2}$) as a subdivision of the unit circle S^1 in the plane. Clearly, every emulation of R_n on $R_{n/2}$ induces a continuous mapping from S^1 to itself. It is well-known that such mappings can be characterized by their topological degree or "winding number". The winding number indicates the number of times the image of S^1 is wrapped around the unit circle when S^1 is traversed once. By analogy we can speak of the winding number of an emulation.

Proposition 5.6.1.1. The winding number of an emulation of R_n on $R_{n/2}$ is -2,-1,o,+1, or +2.

Proof.

Let f be an emulation of R_n on $R_{n/2}$. If the image of R_n wraps around $R_{n/2}$ 3 times or more, then the n nodes of R_n are mapped to a trajectory of at least $\frac{3}{2}$ n consecutive points on $R_{n/2}$. This is impossible. □

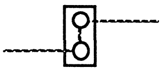
It is relatively straightforward to classify the possible uniform emulations of R_n on $R_{n/2}$ by their (positive) winding number.

Case I. Winding number = o.

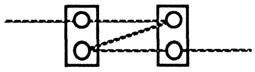
If $f(R_n)$ cannot make a full turn around $R_{n/2}$ then the condition of uniformity forces f to be one of the two forms suggested in table C (a). We shall refer to the emulations as being of "type 1".

Case II. Winding number = 1.

One verifies that f(R) must be composed of a number of "hooks" and "zigzags":



"hook"



"zigzag"

Conversely, any combination of hooks and zigzags defines a uniform emulation of R_n on $R_{n/2}$ with winding number 1. We shall refer to the emulations of this kind as being of "type 2". Table C(b) shows two extreme examples of emulations of type 2.

Case III. Winding number = 2.

$f(R_n)$ is necessarily of the kind suggested in table C(c). We shall refer to the emulations of this kind as being of "type 3".

We conclude the following.

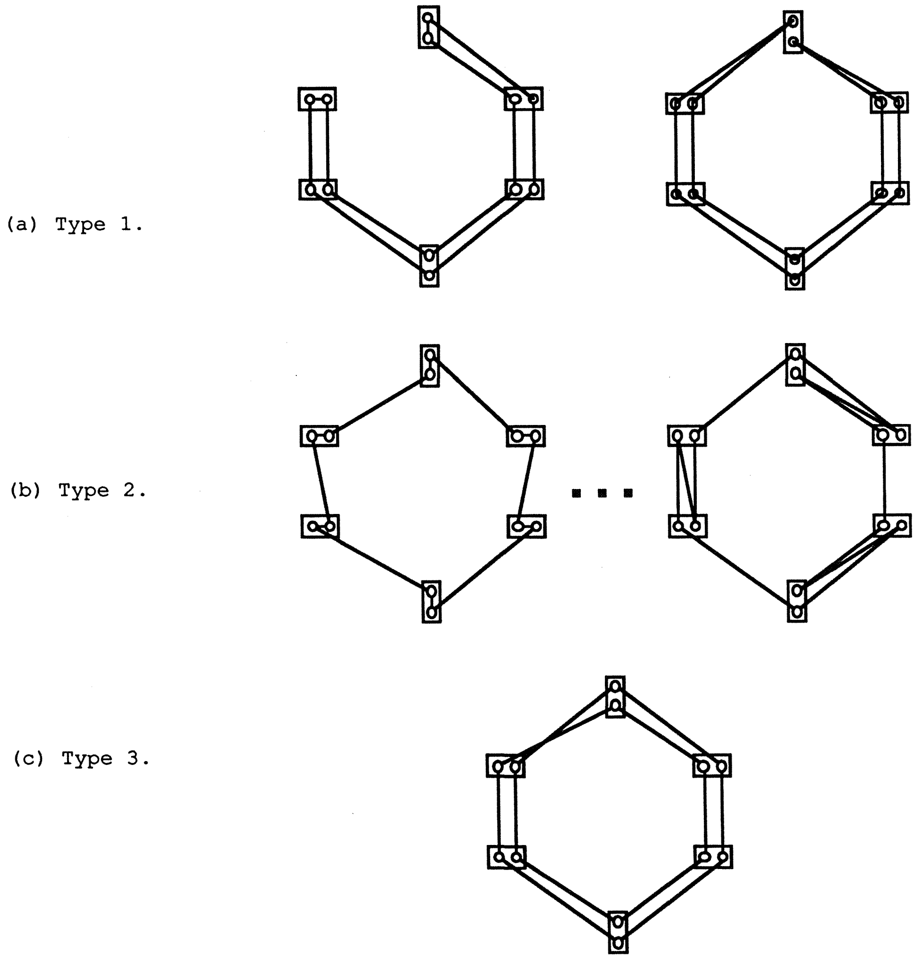
Theorem 5.6.1.2. (Characterization Theorem). For n even, f is a uniform emulation of R_n on $R_{n/2}$ if and only if it is of type 1, type 2, or type 3.

Corollary 5.6.1.3. The number of different uniform emulations of R_n on $R_{n/2}$ is exponential in n.

Proof.

(Two emulations f and g are said to be "different" if g cannot be obtained by a rotational shift of f.) Clearly the number of uniform emulations of R_n on $R_{n/2}$ of type 2 is exponential in n. □

5.6.2. Uniform emulations of GR_n on $GR_{n/2}$. In this section we consider emulations of the grid network (cf. section 1.6.2.) Here we use the



(a) Type 1.

(b) Type 2.

(c) Type 3.

Table C. Classification of the uniform emulations of R_n on $R_{n/2}$ by winding number.

version of the grid with "wrap-around" connections along the boundaries, which gives the nodes a uniform neighborhood structure. By "folding" GR_n, it follows that GR_n can be uniformly emulated on $GR_{n/2}$. Every uniform emulation of GR_n on $GR_{n/2}$ has computation factor 4. The classification of the uniform emulations is presently open, but some useful observations can be made.

As $GR_n \cong R_n \times R_n$, it can effectively be viewed as a torus. Let $n \geq 10$ and let f be a uniform emulation of GR_n on GR_n. Every cycle with 4 nodes, i.e., a "square" in GR_n must be mapped on $GR_{n/2}$ by f in one of the ways shown in Table D.

From this one easily derives that f induces a continuous mapping of the torus to itself. Again the notion of topological degree (winding number) can be introduced, as expounded in homology theory. Let GR_n be "spanned" by the oriented cycles $\vec{a} \cong \{(o,j) \mid o \leq j \leq n-1\}$ and $\vec{b} \cong \{(i,o) \mid o \leq i \leq n-1\}$. A closed curve C can be classified by the pair (k,l), where k is the number of times C is wrapped in the $\vec{a}$-direction and l is the number of times C is wrapped in the $\vec{b}$-direction. One can now classify (uniform) emulations by the topological degrees of $f(\vec{a})$ and $f(\vec{b})$, considered as closed curves on the torus $GR_{n/2}$. The underlying reason is the following fact from homology theory : if closed curves C, C′ on the torus have equal topological degree and f is continuous, then f(C) and f(C′) have equal topological degree on the torus also.

For $n \leq 8$, the same analysis does not necessarily hold. In Table E we give an example of a uniform emulation of GR_6 on GR_3 for which $f(\vec{a}) = f(\{(o,j) \mid o \leq j \leq 5\})$ has topological degree (1,1) and $f(\{(1,j) \mid o \leq j \leq 5\})$ has topological degree (−1,1). (Hence f cannot induce a continuous mapping of the torus to itself.)

Proposition 5.6.2.1. Let f be an emulation of GR_n on $GR_{n/2}$. The topological degree (k,l) of $f(\vec{a})$ and $f(\vec{b})$ satisfies $|k|+|l| \leq 2$.

(The proof follows by observing that the n points of $\vec{a}$ or $\vec{b}$ can be mapped to a trajectory of at most n points on $GR_{n/2}$.)

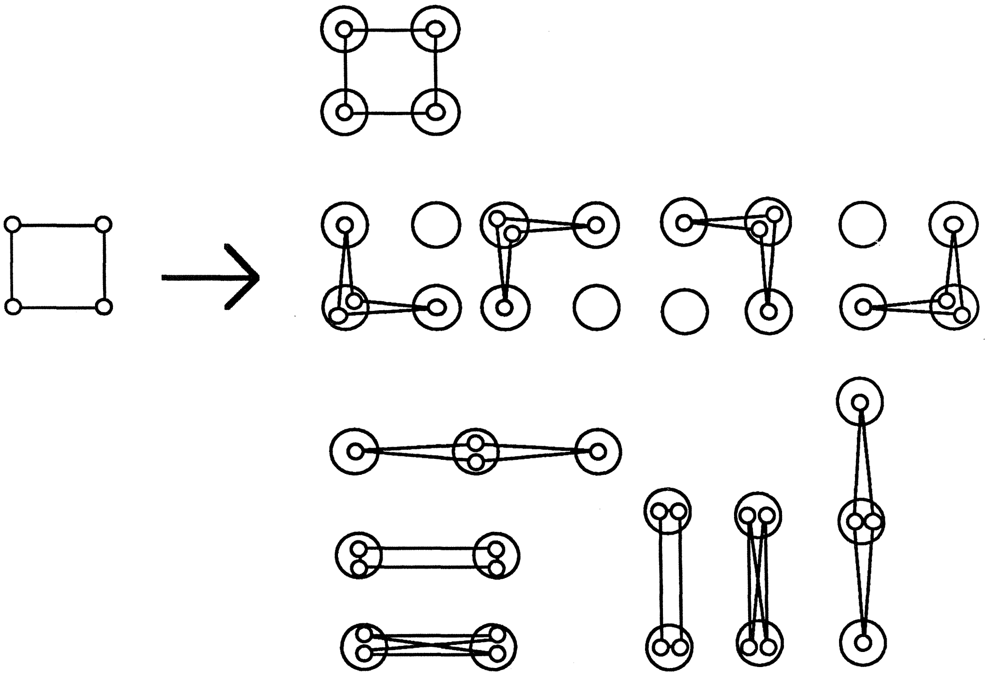


Table D. Possible mappings of a cycle with 4 nodes by an emulation f of GR_n on $GR_{n/2}$, for $n \geq 10$.

23	32	03	13	35	44
34	43	30	31	53	55
01	10	02	14	15	25
24	42	20	41	51	52
00	11	04	12	05	45
22	33	21	40	50	54

Table E. A uniform emulation of GR_6 on GR_3 that does not induce a continuous mapping of the torus to itself.

Theorem 5.6.2.2. The number of uniform emulations of GR_n on $GR_{n/2}$ is at least exponential in n.

Proof.

Let g,h be uniform emulations of R_n on $R_{n/2}$. Clearly the mapping f defined by $f(i,j) = (g(i), h(j))$ is a uniform emulation of GR_n on $GR_{n/2}$. By corollary 5.6.1.3. at least exponentially many uniform emulations are obtained. □

For the uniform emulations f defined in the proof of theorem 5.6.2.2., the topological degrees of $f(\vec{a})$ and $f(\vec{b})$ are of the form (k,o) and (o,1) respectively. Table F shows an example of a uniform emulation f of GR_8 on GR_4 for which $f(\vec{a})$ has topological degree (1,1) and $f(\vec{b})$ has topological degree (1,−1). (The example can easily be generalized to obtain uniform emulations f of GR_n on $GR_{n/2}$ for which $f(\vec{a})$ has topological degree (1,1) and $f(\vec{b})$ has topological degree (1,−1), for all even $n \geq 6$.)

07	32	24	33	14	25	06	15
43	76	60	77	50	61	42	51
22	31	12	23	04	13	05	30
66	75	56	67	40	57	41	74
10	21	02	11	03	36	20	37
54	65	46	55	47	72	64	73
00	17	01	34	26	35	16	27
44	53	45	70	62	71	52	63

Table F. A uniform emulation of GR_8 on GR_4 that is not the direct product of two uniform emulations of R_8 on R_4.

Similar results can be obtained for the d-dimensional analogue of GR_n. Let GR_n^d be the d-dimensional grid network (with wrap-around) with size n in each dimension, i.e., a "hypertorus" with n^d nodes.

<u>Theorem 5.6.2.3</u>. The number of uniform emulations of GR_n^d on $GR_{n/2}^d$ is at least exponential in dn.

The proof is a straightforward extension of the argument used for theorem 5.6.2.2.

CHAPTER SIX

EMULATIONS: FURTHER ANALYSIS

6.1. Introduction. In this chapter we proceed with the analysis of (uniform) emulation. Except for some elementary results, chapter 5 was devoted to results on the characterization of networks on smaller networks (of the same type). In this chapter we give somewhat more diverse results on emulations. In section 6.2. we consider the question of emulating networks of some class C on networks of some (other) class C'. In section 6.3. we show that there is a natural way to describe the networks that can be emulated on a given network H, using a set of permissible emulations. The results lead to interesting characterizations of some networks in terms of their emulated behaviour. In section 6.4. we relate the notion of uniform emulation to the notion of covering, from combinatorial topology, and characterize all possible coverings of networks on (smaller) networks of the same type, for the following classes of networks: ring, grid, cube, complete, cube-connected cycles, 4-pin shuffle and shuffle-exchange networks. In section 6.5. we extend the concept of emulation to the realistic case of networks, that use buses for communication, i.e. connections between arbitrary sets of processors. Several emulations of the spanning bus hypercube and the dual bus hypercube are given.

6.2. Cross emulations. By cross-emulation we refer to the emulation of a network G belonging to some class C_1 on a network H belonging to a different class C_2. The question of cross-emulating G on H can be important if algorithms must be transported from one type of interconnection network to another. We only consider situations with $|G| = |H|$, which means that the resulting uniform (cross-) emulations will necessarily have computation factor 1. Several results of Parker [Pa80] concerning the "topological" equivalence of some common types of multi-stage networks are easily put into this framework. We only consider cross-emulations between S_n, C_n, R_n, and GR_n (as defined in section 1.6.).

Cross-emulations between the shuffle-exchange network and the 4-pin shuffle were studied in section 5.4.2.

In a number of cases the existence of cross-emulations is impossible by degree arguments. For example S_n, C_n, and GR_n cannot be emulated on a ring network of the same number of nodes. C_n and GR_n cannot be emulated on a 4-pin shuffle network with a corresponding number of nodes, and neither can S_n be cross-emulated on the grid network of an equal number of nodes.

Proposition 6.2.1. For $n\geq 2$, S_n cannot be uniformly emulated on C_n.

Proof.

Suppose there was a uniform emulation f of S_n on C_n. Clearly $f(oo^{n-1})$, $f(1o^{n-1})$, and $f(o^{n-1}1)$ must be adjacent to one another in C_n, as the arguments are in S_n. Thus C_n contains a triangle. Contradiction. □

On the positive side, consider GR_{2^n} (with 2^{2n} nodes).

Theorem 6.2.2. For $n\geq 1$, GR_{2^n} can be uniformly emulated on C_{2n}.

Proof.

We prove the following, slightly stronger claim : for every $m,n\geq 1$ there is a uniform emulation of the $2^m \times 2^n$ grid network (with wrap-around connections) on C_{m+n}. Putting m=n proves the theorem. To prove the claim, induct on m and n. For m=n=1 the result is immediate. Assume the claim holds for some $m,n\geq 1$. Let f be a uniform emulation of the $2^m \times 2^n$ grid network on C_{m+n}. Consider the $2^{m+1} \times 2^n$ grid network, and map it to C_{m+n+1} using the mapping f' defined by

$$f'(i,j) = \begin{cases} o.f(i,j) & \text{if } o\leq i<2^m, \\ 1.f(2^{m+1}-i-1,j) & \text{otherwise } (2^m\leq i<2^{m+1}) \end{cases}$$

One easily verifies that f' is a uniform emulation. Likewise the $2^m \times 2^{n+1}$ grid network can be uniformly emulated on C_{m+n+1}. This completes the inductive argument. □

By a degree argument it easily follows that, conversely, C_{2n} can be uniformly emulated on GR_{2^n} only for n=2.

Theorem 6.2.3. For values of n as indicated, R_n can be uniformly emulated on the following networks :

(i) for $n=k^2$, on GR_k.

(ii) for $n=2^k$, on S_k and on C_k.

Proof.

(The results are equivalent to claiming that GR_k, S_k and C_k are hamiltonian.)

(i) Left as an exercise.

(ii) By the existence of binary de Bruijn sequences ([dB46]) it follows that every S_k has a Hamiltonian circuit. To obtain the result for C_k, write $k=k_1+k_2$. As the result is obvious for k=1, we may assume that $k_{1,2}\geq 1$. It is easy to show that R_n can be uniformly emulated on the $2^{k_1} \times 2^{k_2}$ grid network (with wrap-around connections). In the proof of theorem 6.2.2. it was shown that the $2^{k_1} \times 2^{k_2}$ grid network can be uniformly emulated on $C_{k_1+k_2}=C_k$. By transitivity the result follows. □

Observe that every uniform emulation of the ring of 2^k elements on C_k corresponds to a Gray code of length k (cf. [RND77]).

6.3. Defining networks by emulation. Every network $H = (V_H,E_H)$ can act as a "host" under emulation for many different, larger networks. If we restrict the class of admissible (uniform) mappings that should act as emulations, then the set of graphs G that can be emulated on H will likely be restricted also. Our main result will be that S_n, C_n, R_n and GR_n are uniquely defined by their emulations on S_{n-1}, C_{n-1}, $R_{n/2}$ and $GR_{n/2}$ respectively. In section 6.3.1. we derive some general results on defining networks by admissible sets of emulations. In section 6.3.2. we prove the main results.

6.3.1. General characterizations. Let $H = (V_H, E_H)$ be a given host network, V a set of nodes with $|V| \geq |V_H|$, and F a collection of functions from V onto V_H.

Definition. A network $G = (V,E)$ is said to be F-emulated on H if every $f \in F$ is an emulation of G on H.

Our aim will be to characterize all networks G that are F-emulated on H, given F and H. We assume H and V to be fixed, and F to be variable.

Definition. $E_F = \{ (v,v') \mid v,v' \in V,\ v \neq v' \text{ and } \forall_{f \in F} : f(v) = f(v') \text{ or } (f(v),f(v')) \in E_H \}$.

Theorem 6.3.1.1. (Characterization Theorem) G is F-emulated on H if and only if G is a spanning subgraph of (V,E_F).

Proof.

Let $G = (V,E)$ be F-emulated on H, and let $(v,v') \in E$. By definition (of emulation) we have that for every $f \in F$: $f(v) = f(v')$ or $(f(v),f(v')) \in E_H$. Thus $(v,v') \in E_F$. It follows that $E \subseteq E_F$, and G is a spanning subgraph of (V,E_F). Conversely, it is clear that (V,E_F) is F-emulated on H by definition. Clearly, every spanning subgraph is F-emulated on H also. □

It follows that (V,E_F) is the maximal graph that can be F-emulated on H. Define $f : V \rightarrow V_H$ to be uniform if for all $h \in V_H$: $|f^{-1}(h)| = c$, for some constant $c = |V|/|V_H|$ (the computation factor).

Theorem 6.3.1.2. Let $f,f' : V \rightarrow V_H$ be uniform functions. Then $(V,E_{\{f\}})$ and $(V,E_{\{f'\}})$ are isomorphic graphs.

Proof.

Since $f,f' : V \rightarrow V_H$ are uniform (and thus map equal sized piles of elements onto every node of H) there exists a permutation $\Pi : V \rightarrow V$ such that for all $v \in V$: $f(v) = f'(\Pi(v))$. One easily verifies that Π is, in fact, an isomorphism of $(V,E_{\{f\}})$ and $(V,E_{\{f'\}})$. □

We derive a further result to characterize $(V,E_{\{f\}})$ when f is uniform. Let c be as defined above.

Lemma 6.3.1.3. Let $f : V \rightarrow V_H$ be uniform. Let d_{out} and d_{in} be the maximum out-degree and the maximum in-degree of the nodes in H, respectively. (If H is undirected, let $d_{out} = d_{in}$ be the maximum degree in H.)

(i) the maximum out-degree in $(V,E_{\{f\}})$ equals $(d_{out}+1)\cdot c-1$.

(ii) the maximum in-degree in $(V,E_{\{f\}})$ equals $(d_{in}+1)\cdot c-1$.

(iii) If G and H are undirected graphs, then $|E_{\{f\}}| = \frac{1}{2}c(c-1)|V_H| + c^2\cdot|E_H|$.

(iv) If G and H are directed graphs, then $|E_{\{f\}}| = c(c-1)|V_H| + c^2\cdot|E_H|$.

Proof.

(i) Consider any node $v \in V$. By uniformity there are precisely c-1 nodes $v'\neq v$ with $f(v) = f(v')$, which thus accounts for c-1 outgoing edges with this property. Next there are at most $d_{out}.c$ nodes v' with $(f(v),f(v')) \in E_H$. This accounts for a maximal out-degree of $c-1+d_{out}.c = (d_{out}+1)c-1$. By choosing v such that f(v) has maximum out-degree, it is clear that the bound is attained.

(ii) Similar to (i).

(iii) E_H contains $|V_H|\cdot\frac{1}{2}c(c-1)$ edges (v,v') with $f(v) = f(v')$, because c nodes of V are mapped to every $h \in V_H$. Every edge $(h,h') \in E_H$ accounts for c^2 edges (v,v') with $f(v) = h$ and $f(v') = h'$. By definition E_F contains no other edges than the ones that were distinguished.

(iv) Similar to (iii). □

Lemma 6.3.1.3. will be useful later because, whenever $f \in F$ and G is F-emulated on H, then G is a spanning subgraph of $(V,E_{\{f\}})$.

6.3.2. Characterization of the 4-pin shuffle, the cube, the ring and the grid networks by emulation. We use the definitions and results concerning S_n, C_n, R_n and GR_n as presented in section 1.6. First we consider S_n, the 4-pin shuffle graph on 2^n nodes. From table A we recall the following uniform emulations of S_n on S_{n-1} :

f_1 : $f_1(b_1..b_n) = b_1..b_{n-1}$,
f_2 : $f_2(b_1..b_n) = b_2..b_n$,
f_3 : $f_3(b_1..b_n) = c_1..c_{n-1}$ with $c_i = (b_i \equiv b_{i+1})$ for $1 \le i \le n-1$

We show that S_n is uniquely characterized by these three emulations on S_{n-1}. Let $V \cong V_n$ $(=(\frac{0}{1})^n)$, $n \ge 2$.

Theorem 6.3.2.1. $(V, E_{\{f_1,f_2,f_3\}}) = S_n$.
Proof.

Clearly S_n is a spanning subgraph of $(V, E_{\{f_1,f_2,f_3\}})$, by definition (or theorem 6.3.1.1.). We show that every edge of $(V, E_{\{f_1,f_2,f_3\}}$ must be an edge of $S_n = (V_n, E_n)$. Consider any edge $(b_1..b_n, c_1..c_n) \in E_{\{f_1,f_2,f_3\}}$. We distinguish the following cases:

(a) $f_i(b_1..b_n) = f_i(c_1..c_n)$ for $1 \le i \le 2$. It follows that $b_1..b_{n-1} = c_1..c_{n-1}$ and $b_2..b_n = c_2..c_n$ and (hence) $b_1..b_n = c_1..c_n$, contradicting that we had an edge between distinct points.

(b) $f_i(b_1..b_n) = f_i(c_1..c_n)$ for $i = 1,3$ (and $(f_2(b_1..b_n), f_2(c_1..c_n)) \in E_{n-1}$). It follows that $b_1..b_{n-1} = c_1..c_{n-1}$ and $(b_{n-1} \equiv b_n) = (c_{n-1} \equiv c_n)$, so $b_1..b_n = c_1..c_n$, a contradiction.

(c) $(f_1(b_1..b_n), f_1(c_1..c_n)) \in E_{n-1}$ and $f_2(b_1..b_n) = f_2(c_1..c_n)$. It follows that $b_2..b_{n-1}\alpha = c_1..c_{n-1}$ and $b_2..b_n = c_2..c_n$ for some α, hence $b_1..b_n = b_1\alpha^{n-1}$ and $c_1..c_n = \alpha^n$. Clearly $(b_1\alpha^{n-1}, \alpha^n) \in E_n$.

(d) $f_1(b_1..b_n) = f_1(c_1..c_n)$ and $(f_i(b_1..b_n), f_i(c_1..c_n)) \in E_{n-1}$ for $i = 2,3$. It follows that $b_1..b_{n-1} = c_1..c_{n-1}$ and $b_3..b_n\alpha = c_2..c_n$ for some α, hence $b_1..b_n = b_1b_n^{n-1}$ and $c_1..c_n = b_1b_n^{n-2}\alpha$. Now $(f_3(b_1b_n^{n-1}), f_3(b_1b_n^{n-2}\alpha)) \in E_{n-1}$ implies $b_1 = b_n$, and clearly $(b_1b_n^{n-1}, b_1b_n^{n-2}\alpha) \in E_n$.

(e) $(f_i(b_1..b_n), f_i(c_1..c_n)) \in E_{n-1}$ for $i = 1,2$. It follows that $b_2..b_{n-1}\alpha = c_1..c_{n-1}$ and $b_3..b_n\beta = c_2..c_n$ for suitable α and β, hence $b_1..b_n = b_1c_1..c_{n-1}$. Clearly $(b_1c_1..c_{n-1}, c_1..c_n) \in E_n$. □

(It can be verified that no subset of $\{f_1, f_2, f_3\}$ is sufficient to characterize S_n.) Next consider C_n, the cube network on 2^n nodes. We select the following uniform emulations of C_n on C_{n-1}:

f_1 : $f_1(b_1..b_n) = b_1..b_{n-1}$

f_4 : $f_4(b_1..b_n) = (b_1 \equiv b_2)b_3..b_n$

Theorem 6.3.2.2. For $n \geq 3$, $(V,E_{\{f_1,f_4\}})=C_n$.

Proof.

Clearly C_n is a spanning subgraph of $(V,E_{\{f_1,f_4\}})$. Consider any edge $(b_1..b_n,c_1..c_n) \in E_{\{f_1,f_4\}}$. We distinguish the following cases :

(a) $f_1(b_1..b_n) = f_1(c_1..c_n)$. It follows that $b_1..b_{n-1} = c_1..c_{n-1}$, and either $b_1..b_n = c_1..c_n$ (a contradiction) or $b_1..b_n = c_1..c_{n-1}c_n$. It follows that $(b_1..b_n,c_1..c_n) \in E_n$.

(b) $(f_1(b_1..b_n),f_1(c_1..c_n)) \in E_{n-1}$ and $f_4(b_1..b_n) = f_4(c_1..c_n)$. It follows that $d(b_1..b_{n-1},c_1..c_{n-1}) = 1$ and $b_1..b_n = b_1b_2c_3..c_n$ with $(b_1 \equiv b_2) = (c_1 \equiv c_2)$. It follows that $b_n = c_n$, and thus $(b_1..b_n,c_1..c_n) \in E_n$.

(c) $(f_1(b_1..b_n),f_1(c_1..c_n)) \in E_{n-1}$ and $(f_4(b_1..b_n),f_4(c_1..c_n)) \in E_{n-1}$. It follows that $d(b_1..b_{n-1},c_1..c_{n-1}) = 1$ and $d(\alpha b_3..b_n,\beta c_3..c_n) = 1$, with $\alpha = (b_1 \equiv b_2)$ and $\beta = (c_1 \equiv c_2)$. If $\alpha=\beta$ then necessarily $b_1b_2 = c_1c_2$ and $d(b_1..b_n,c_1..c_n) = 1$ thus $(b_1..b_n,c_1..c_n) \in E_n$. If $\alpha \neq \beta$ then $b_3..b_n = c_3..c_n$ and (hence) $d(b_1b_2,c_1c_2) = 1$. Clearly it follows that $(b_1..b_n,c_1..c_n) \in E_n$.

We conclude that every edge of $(V,E_{\{f_1,f_4\}})$ also is an edge of C_n. □

Theorem 6.3.2.2. is "minimal" in the sense that C_n cannot be characterized from C_{n-1} by means of just one uniform emulation.

Proposition 6.3.2.3. There does not exist a uniform emulation f of C_n on C_{n-1} such that $(V,E_{\{f\}}) = C_n$.

Proof.

Observe that (the undirected graph) C_{n-1} has 2^{n-1} nodes and $\frac{1}{2}.2^{n-1}(n-1)$ edges. Suppose a uniform mapping $f : V \rightarrow V_{n-1}$ exists with $(V,E_{\{f\}}) = C_n$. By lemma 6.3.1.3. (iii) $(V,E_{\{f\}})$ must have $n.2^n-2^{n-1}$ edges (c=2), which is more than C_n can have. □

Consider R_n, the ring on n nodes. Define the following uniform emulations of R_n on $R_{n/2}$ (n even) :

$g_1 : g_1(i) = \lfloor \frac{i}{2} \rfloor$
$g_2 : g_2(i) = (i \bmod n/2)$

Theorem 6.3.2.4. For $n>8$, $(V,E_{\{g_1,g_2\}}) = R_n$.

Proof.

Clearly R_n is a spanning subgraph of $(V,E_{\{g_1,g_2\}})$. Consider any edge $(i,j) \in E_{\{g_1,g_2\}}$. If $g_1(i) = g_1(j)$ then $|i-j|=1$ and $(i,j) \in E_n$. If $(g_1(i),g_1(j)) \in E_{n/2}$, then we may assume without loss of generality that $\lfloor i/2 \rfloor = \lfloor j/2 \rfloor+1$ (mod $n/2$). It follows that $i \equiv j+2+\delta_i-\delta_j$ (mod n), with δ_i and δ_j denoting $i \bmod 2$, and $j \bmod 2$, respectively. Now, in addition, $g_2(i) = g_2(j)$ or $(g_2(i), g_2(j)) \in E_{n/2}$. If $g_2(i) = g_2(j)$ then $|i-j| \equiv o$ (mod $n/2$), hence $2+\delta_i-\delta_j \equiv o$ (mod $n/2$) and, by the assumption on n, necessarily $2+\delta_i-\delta_j = o$ and $i=j$. Contradiction. If $(g_2(i),g_2(j)) \in E_{n/2}$ then $i \equiv j \pm 1$ (mod $n/2$), hence $2+\delta_i-\delta_j \equiv \pm 1$ (mod $n/2$). Since $n>8$, we have $2+\delta_i-\delta_j = 1$ and $i \equiv j+1$ (mod n). Thus $(i,j) \in E_n$. We conclude that every edge of $(V,E_{\{g_1,g_2\}})$ is an edge of R_n. □

Finally consider GR_n, the grid network on n^2 nodes. Define the following uniform emulations of GR_n on $GR_{n/2}$ (n even) :

$h_1 : h_1(i,j) = (\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor)$
$h_2 : h_2(i,j) = (i \bmod n/2,\ j \bmod n/2)$

Theorem 6.3.2.5. For $n>8$, $V,E_{\{h_1,h_2\}}) = GR_n$.

Proof.

Similar to the proof of theorem 6.3.2.4. □

6.4. The classification of coverings of processor networks.

6.4.1. Introduction. The concept of covering is a fundamental notion in combinatorial topology. Every covering is a mapping of a d-dimensional complex onto (another) d-dimensional complex. The notion of a one-dimensional complex corresponds with the notion of an undirected graph (with possibly parallel edges and self-loops). (Throughout this section we will use the terminology of graphs.) In this section we will study the coverings of graphs, for graphs representing the interconnection structure of processor networks. The following definition can be found in [Re32], and in many other standard books on combinatorial topology, often in generalized form.

Definition. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be undirected graphs. We say that $f: V_G \rightarrow V_H$ covers G on H (or "G covers H") iff

1. f is surjective
2. $(v,w)\in E_G$ implies $((f(v), f(w)) \in E_H$
3. For all $v\in V_G$, let $w_1,\ldots,w_{k_v}$ be the neighbors of v in G. Then $f(w_1),\ldots,f(w_{k_v})$ are all different and each neighbor v' of f(v) is equal to a $f(w_i)$ $(1\le i\le k_v)$.

Theorem 6.4.1.1. [Re32] Every covering of a graph G on a connected graph H is uniform.

Proof. Suppose f is a covering of G on H that is not uniform, and H is connected. Then there must be adjacent nodes v,w with $|f^{-1}(v)|\neq|f^{-1}(w)|$. This contradicts with the definition of covering. □

Corollary 6.4.1.2. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be undirected graphs and let H be connected. Every covering of G on H is a uniform emulation of G on H.

In [Re32] the following method is given to describe all graphs that cover a given connected graph H with computation factor c:

- determine a spanning tree B of H,
- make c copies of B,
- for every edge in H that is not an edge in the tree B we have c copies of both its endpoints. These copies are connected on a one-to-one basis: every copy of first endpoint must be connected to a unique copy of the second endpoint. (This gives c! manners to make the connections.)

Up to isomorphism every graph G that covers H can be obtained in the described manner. For further details of the theory of coverings of graphs and more-dimensional complexes, see e.g. [Re32].

We introduce a directed version of the notion of covering:

Definition. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be directed graphs. We say that $f\colon V_G \rightarrow V_H$ covers G on H (or "G covers H") iff

1. f is surjective
2. $(v,w) \in E_G$ implies $(f(v), f(w)) \in E_H$.
3. For all $v \in V_G$, let $w_1,..,w_{k_v}$ be the successors of v in G. Then $f(w_1),..,f(w_{k_v})$ are all different, and each successor v' of f(v) is equal to a $f(w_i)$ $(1 \le i \le k_v)$.
4. For all $v \in V_G$, let $w'_1,..,w'_{l_v}$ be the predecessors of v in G. Then $f(w'_1),..,f(w'_{l_v})$ are all different, and each predecessor v' of f(v) is equal to a $f(w'_i)$ $(1 \le i \le l_v)$.

Let $\overline{G}$ be the undirected graph obtained by ignoring the direction of the edges in the directed graph $G=(V,E)$, i.e. $\overline{G}=(V, \overline{E})$, with $\overline{E}=\{(v,w) \mid (v,w) \in E \text{ or } (w,v) \in E\}$.

Lemma 6.4.1.3. Let G,H be directed graphs. If G covers H, then $\overline{G}$ covers $\overline{H}$.

Proof. Clear from the definitions. □

Proposition 6.4.1.4. Every covering of a directed graph G on a directed, connected graph H is a uniform emulation.

The following theorem from combinatorial topology characterizes precisely which graphs cover a given tree network.

Theorem 6.4.1.5. [Re32] Let H be a tree, and let G cover H. Then G consists of a number (≥1) of connected components, each graph isomorphic to H. In particular, if G is connected then G and H are graph isomorphic.

Similar to the description of the graphs that cover an undirected graph H with computation factor c, one can describe all graphs that cover a directed graph $H=(V_H,E_H)$ with computation factor c:

- let $B=(V_H,E_B)$ be a directed graph, such that:

  (i) $\overline{B}$ is a spanning tree of $\overline{H}$

(ii) $|E_B|=|V_H|-1$, so for every pair of nodes $v_1,v_2 \in V_H$ at most one edge between these nodes (regardless to the direction) is an edge in E_B.

- make c copies of B
- for every edge $(v_1,v_2) \in E_H$ that is not an edge in E_B we have c copies of both its endpoints v_1,v_2. These copies are connected on a one-to-one basis: every copy of the first endpoint must be connected to a unique copy of the second endpoint: the direction of the edges is from a copy of v_1 to a copy of v_2.

Up to isomorphism every graph G that covers H can be obtained in the described manner. (The proof of this characterization uses lemma 6.4.1.5. and the result for the undirected case, and is omitted.)

The notion of covering of processor networks also appears in a different area of distributed computing. Angluin [An80] considered the problem of finding a leader in processor networks (cf. chapter 2), where processors do not have their own unique identity number, (i.e. processors having an equal degree are identical), and proved that no algorithm for this problem can work "correctly" for both graphs G and H, where G and H are not isomorphic and H is a covering of G. For the precise assumptions we refer to [An80]. The result can be generalized for directed graphs, by using a similar proof and the directed version of the notion of covering that was introduced in this section.

The notion of covering is much more restricted than the notion of uniform emulation: in many cases there are many more uniform emulations possible of some network G on some other network H then there are coverings possible, and in some cases there exist uniform emulations but no coverings of G on H. The classification of coverings of networks on smaller networks of the same type is a special case of the problem of classification of uniform emulations of networks on smaller networks of the same type. In this section the classification of coverings is carried out for the following types of graphs; each type is used or proposed as the interconnection structure for current processor networks:

- the ring	(section 6.4.2.1.)
- the grid	(section 6.4.2.2.)
- the cube	(section 6.4.2.3.)
- complete networks	(section 6.4.2.4.)
- the cube connected cycles	(section 6.4.3.)
- the 4-pin shuffle	(section 6.4.4.)
- the shuffle-exchange network	(section 6.4.5.)

6.4.2. Coverings of ring, grid, cube and complete networks.

6.4.2.1. Ring networks. Recall the definition of the ring network from section 1.6.1. It is easy to classify the coverings of the ring networks. The following propositions are given without proof.

Proposition 6.4.2.1.1. Let G be a graph, consisting of connected components $G_1,\dots,G_j$ and let $k \in N^+$, $k\geq 3$. G covers R_k if and only if for all i, $1\leq i\leq j$ G_i is graph isomorphic to a ring R_n with $k|n$.

Proposition 6.4.2.1.2. Let H be a graph, and $n\in N^+$. R_n covers H if and only if H is graph isomorphic to a ring R_k with $k|n$.

Proposition 6.4.2.1.3. The coverings of R_n on R_k with $k\geq 3$, $k|n$ are given by the following list:

(i) $f_j(m)$ = (m+j) mod k (for j, $0\leq j<k$)

(ii) $\bar{f}_j(m)$ = (−m+j) mod k (for j, $0\leq j<k$)

Proof. It is easy to verify that every f_j, $\bar{f}_j$ is a covering of R_n on R_k. Let f be a covering of R_n on R_k. Consider f(0) and f(1). If f(1)=(f(0)+1) mod k, then f is of type f_j; if f(1)=(f(0)−1) mod k, then f is of type $\bar{f}_j$. In both cases j=f(0). With induction one can verify that f must be of the designed type. □

Corollary 6.4.2.1.4. For all k,n, k|n, $k\geq 3$ there are precisely 2k coverings of R_n on R_k.

6.4.2.2. Grid networks. Recall the definitions of the two-dimensional grid networks with, and without boundary connections from section 1.6.2.

Proposition 6.4.2.2.1. If $n \neq m$ then there exist no covering of GR'_n on GR'_m.

Proof. GR'_n and GR'_m both have exactly 4 nodes of degree 2. Because degree(v) = degree(f(v)) for all $v \in V_n$, and a covering of GR'_n or GR'_m must be uniform, with a computation factor $\neq 1$ because $n \neq m$, there exists no covering of GR'_n or GR'_m. □

One can view the two-dimensional grid network with boundary connections as the product of two ring networks: $GR_n \cong R_n \times R_n$. Every covering of GR'_n or GR'_m ($m|n$, $m \geq 5$) can also be written as the product of two coverings of R_n on R_m.

Theorem 6.4.2.2.2. Let $m|n$, $m \geq 5$, and let f be a function $V_n \to V_m$. f is a covering of GR_n on GR_m, if and only if there are coverings f_1, f_2 of R_n on R_m, with

$$f((i,j)) = (f_1(i),\ f_2(j)) \text{ for all } (i,j) \in V_n$$

$$\text{or } f((i,j)) = (f_1(j),\ f_2(i)) \text{ for all } (i,j) \in V_n.$$

Proof. First note that every cycle with 4 nodes (a "square") in GR_n must be mapped on a cycle with 4 nodes (a "square") in GR_m. With induction one can prove that, after having fixed the image of one cycle with 4 nodes (square) in GR_n, the whole covering is fixed and necessarily of the described form. □

By the results of section 6.4.2.1. theorem 6.4.2.2.2. implies a complete characterization of the coverings of GR_n on GR_m, $m|n$, $m \geq 5$.

6.4.2.3. Cube networks. Recall the definition of the cube network from section 1.6.4.

Proposition 6.4.2.3.1. Let C_n cover C_m. Then $n=m$.

Proof.

This follows because degree $(f(v))$=degree(v), for any covering of C_n on C_m and $v \in C_n$. □

6.4.2.4. Complete networks. The same argument can be used for complete networks.

Proposition 6.4.2.4.1. Let K_n cover K_m. Then $n=m$.

6.4.3. Coverings of the cube connected cycles. In this section we consider coverings of the cube connected cycles. Recall the definition of the cube connected cycles network from section 1.6.5. Edges of the type $(p_1 \ldots p_{i-1} | p_i \ldots p_r,\ p_1 \ldots \overline{p_{i-1}} | p_i \ldots p_r)$ are called exchange edges; edges of the type $(p_1 \ldots p_{i-1} | p_i \ldots p_r,\ p_1 \ldots p_i | p_{i+1} \ldots p_r)$ are called divide shift edges. Notice that every node is adjacent to 2 divide shift edges and one exchange edge. When a divide shift edge is passed in the direction from $p_1 \ldots p_{i-1} | p_i \ldots p_r$ to $p_1 \ldots p_i | p_{i+1} \ldots p_r$ we say it is passed in the positive direction, otherwise we say it is passed in the negative direction.

We will only consider coverings of CCC_r on CCC_s with $r \geq s \geq 9$.

Lemma 6.4.3.1. Every cycle in CCC_r with $r \geq 9$ has at least 8 nodes. For every cycle with exactly 8 nodes in CCC_r with $r \geq 9$ there are $p_1 \ldots p_r \in \binom{0}{1}^r$, $i \in \{1,..,r\}$, such that the successive nodes visited by the cycle (assuming starting point and direction in which the cycle is traversed are well chosen) are:

$$\begin{array}{l}
p_1 \cdots p_i | \ p_{i+1} \ p_{i+2} \cdots p_r \\
p_1 \cdots p_i \ \underline{p_{i+1}} | \ p_{i+2} \cdots p_r \\
p_1 \cdots p_i \ \underline{p_{i+1} |} \ p_{i+2} \cdots p_r \\
p_1 \cdots \underline{p_i} | \ \underline{p_{i+1}} \ p_{i+2} \cdots p_r \\
p_1 \cdots \underline{p_i} | \ \underline{p_{i+1}} \ p_{i+2} \cdots p_r \\
p_1 \cdots \underline{p_i} \ p_{i+1} | \ p_{i+2} \cdots p_r \\
p_1 \cdots \underline{p_i} \ p_{i+1} | \ p_{i+2} \cdots p_r \\
p_1 \cdots p_i | \ p_{i+1} \ p_{i+2} \cdots p_r
\end{array}$$

Proof.

This is easy, but tedious to check. □

Now let f be a covering of CCC_r or CCC_s, with $r \geq s \geq 9$. Every cycle with 8 nodes of CCC_r must be mapped upon a cycle with 8 nodes of CCC_s. Notice that every edge that is adjacent to a cycle with 8 nodes but not part of this cycle, necessarily is a divide shift edge.

Lemma 6.4.3.2. Let f cover CCC_r on CCC_s, $r \geq s \geq 9$. Then f maps nodes connected by a divide shift edge on nodes connected by a divide shift edge and nodes connected by an exchange edge on nodes connected by an exchange edge.

Proof.

For every divide shift edge e there is a cycle of 8 nodes to which e is adjacent, but of which e is no part. Hence e must be mapped upon an edge, adjacent to a cycle of 8 nodes, that is: a divide shift edge. Because every node is adjacent to exactly 2 divide shift edges and one exchange edge, exchange edges must be mapped upon exchange edges. □

Now let f(o|o..o) be given. There are two possible values for f(oo|o..o), the node reachable from f(o|o..o) by a divide shift in positive direction, and the node reachable from f(o|o..o) by a divide shift in negative direction. The choice of f(o|o..o) and f(oo|o..o) fixes the covering f.

Lemma 6.4.3.3. Let b and c be nodes in CCC_r, that are connected by a divide shift edge. There is at most one covering f of CCC_r on CCC_s ($r \geq s \geq 9$) with f(o|o..o) = b and f(oo|o..o) = c.

Proof.

Suppose f(o|o...o) = b = $p_1 \ldots p_{i-1} | p_i \ldots p_s$ and f(oo|o...o) = c = $p_1 \ldots p_{i-1} | p_{i+1} \ldots p_s$. (If b→c is a divide shift in negative direction, then the proof is analogous.) Let f be a covering of CCC_r on CCC_s. If $f(q_1 \ldots q_{j-1} | q_j \ldots q_r)$ reaches $f(q_1 \ldots q_j | q_{j+1} \ldots q_r)$ by a divide shift in positive direction, then every $f(q_1 \ldots q_{j'-1} | q_{j'} \ldots q_r)$ reaches

$f(q_1...q_{j'}|q_{j'+1}...q_r)$ by a divide shift in positive direction ($1 \le j' \le r$). Similarly for negative directions.

Proposition 6.4.3.3.1. Let b',c' be nodes in CCC_r, and c' can be reached from b' by a divide shift in positive direction. Then f(c') can be reached from f(b') by a divide shift in a positive direction.

Proof.

Suppose not. We now can divide $(\frac{o}{1})^r$ in two disjunctive sets A and B, with $A = \{q_1...q_r \mid f(q_1...q_{i-1}|q_i...q_r)$ reaches $f(q_1...q_i|q_{i+1}...q_r)$ by a divide shift in positive direction$\}$ and $B = \{q_1...q_r \mid f(q_1...q_{i-1}|q_i...q_r)$ reaches $f(q_1...q_i|q_{i+1}...q_r)$ by a divide shift in negative direction$\}$. (Note that the choice of i is not important.) We know that A and B are not empty. For instance $o^n \in A$.

There must be strings $a \in A$, $b \in B$, that differ in exactly one bit-position. So there are $p_1...p_r \in (\frac{o}{1})^r$, i, $1 \le i \le r$, $q_1...q_s \in (\frac{o}{1})^s$, $q'_1...q'_s \in (\frac{o}{1})^s$, j, $1 \le j \le s$, j', $1 \le j' \le s$, with

- $f(p_1...p_{i-1}|\ p_i\ p_{i+1}...p_r) = q_1...q_{j-1}|\ q_j\ q_{j+1}...q_r$
- $f(p_1...p_{i-1}\ \underline{p_i}|\ p_{i+1}...p_r) = q'_1...q'_{j-1}\ q'_j|\ q'_{j+1}...q_r$,
- $f(p_1...p_{i-1}|\ \underline{p_i}\ p_{i+1}...p_r) = q'_1...q'_{j'-1}q'_{j'}|\ q'_{j'+1}...q'_r$
- $f(p_1...p_{i-1}\ \overline{p_i}|\ p_{i+1}...p_r) = q_1...q_{j'-1}|q_{j'}q_{j'+1}...q_r$

From 6.4.3.2. we have

- $f(p_1...p_{i-1}\ \overline{\underline{p_i}}|\ p_{i+1}...p_r) = q_1...q_{j-1}\ \overline{\underline{q_j}}|\ q_{j+1}...q_r$, hence
- $f(p_1...p_{i-1}|\ p_i\ p_{i+1}...p_r) = q_1...q_{j-1}\ q_j\ q_{j+1}|q_{j+2}...q_r$.

Now notice that $p_1...p_{i-1}|p_ip_{i+1}...p_r$; $p_1...p_{i-1}p_i|p_{i+1}...p_r$; $p_1...p_{i-1}|\overline{p_i}p_{i+1}...p_r$; $p_1...p_{i-1}p_i|p_{i+1}...p_r$ are 4 successive nodes in a cycle with 8 nodes in CCC_r. However, their images are not 4 successive nodes in a cycle with 8 nodes in CCC_s. So there is a cycle with 8 nodes that is not mapped upon a cycle with 8 nodes. Contradiction. □

This shows that f is completely determined by the choice of f(o|o..o) and f(oo|o..o). Let d be a node of CCC_r. Look at the path from o|o..o to d, and the f-images of the nodes on this path. If the path uses an exchange edge, a divide shift in positive direction or a divide shift in negative direction, respectively, then the path formed by the f-images must use the same type of edge. Hence f(d) is fully determined by f(o|o..o) and f(oo|o..o). □

Corollary 6.4.3.4. Let $s|r$. There are at most $2s \cdot 2^s$ coverings of CCC_r on CCC_s.

Proof.

There are $s \cdot 2^s$ possible choices for f(o|o..o). For f(oo|o..o) there are 2 choices left. □

Lemma 6.4.3.5. Let CCC_r cover CCC_s. Then $s|r$.

Proof.

Our analysis shows that cycles, consisting of only divide-shift edges in CCC_r must be mapped upon cycles, consisting of only divide-shift edges in CCC_s. Hence cycles with r nodes must cover cycles with s nodes, so $s|r$. □

Consider the following graph isomorphisms of CCC_s.

1. Let $I \subseteq \{1,..,s\}$. The function X_I does not move the divide, but flips the bits with index in I:

   $X_I(p_1 \ldots p_j | p_{j+1} \ldots p_s) = q_1 \ldots q_{j'} | q_{j'+1} \ldots q_s \Leftrightarrow$
   $j=j'$; $q_i=p_i$ for all $i \notin I$; $q_i \neq p_i$ for all $i \in I$.

2. Let $t \in \{0,..,s-1\}$. The function S_t shifts the whole string, the divide inclusive, t positions to the left:

   $S_t(p_1 \ldots p_i | p_{i+1} .. p_s) = p_{t+1} .. p_i | p_{i+1} .. p_s \; p_1 \ldots p_t$
   (or, of course, $p_{t+1} \ldots p_s p_1 \ldots p_i | p_{i+1} \ldots p_t$).

3. The function R reverses the string. Notice that the divide is still placed after the same bit, not between the same bits, as originally:

   $R(p_1 \ldots p_i | p_{i+1} \ldots p_s) = p_s \ldots p_{i+1} p_i | p_{i-1} \ldots p_1$.

It is easy to check that for every $I \subseteq \{1,..,s\}$, $t \in \{0,..,s-1\}$, X_I, S_t and R are graph isomorphisms (= coverings) of CCC_s onto itself.

Corollary 6.4.3.6. Every graph isomorphism of CCC_s is of one of the following forms:

$$X_I \circ S_t \quad (I \subseteq \{1,..,s\},\ o \leq t \leq s-1)$$

$$\text{or } X_I \circ S_t \circ R \quad (I \subseteq \{1,..,s\},\ o \leq t \leq s-1)$$

Proof.

The described forms give $2s.2^s$ different graph isomorphisms of CCC_s. Corollary 6.4.3.4. shows there cannot be more (every isomorphism is a covering). □

Consider the following function F: $CCC_r \rightarrow CCC_s$:

$$F(p_1 \ldots p_i | p_{i+1} \ldots p_r) = q_1 \ldots q_{i'} | q_{i'+1} \ldots q_s$$

$$\Leftrightarrow q_j = \left(\sum_{k=0}^{r/s-1} b_{ks+j} \right) \bmod 2 \text{ (for all } j,\ 1 \leq j \leq s)$$

and i'=i mod s (i.e. the divide is placed after the i mod s^{th} bit).

Lemma 6.4.3.7. F is a covering of CCC_r on CCC_s.

Proof.

It is clear that F is surjective.

Consider a node $p_1 \ldots p_i | p_{i+1} \ldots p_r$ in CCC_r and its 3 neighbors. From the definition of F it is clear that each of these neighbors is mapped upon another neighbor of F(p). □

Theorem 6.4.3.8. Every covering f of CCC_r on CCC_s can be written as $f = J \circ F$, where F is given by the definition above, and J is a graph isomorphism of CCC_s onto itself.

Proof.

Every function of the form $J \circ F$ is a covering of CCC_r on CCC_s. There are $2s \cdot 2^s$ possible choices for J, so there are $2s \cdot 2^s$ possible functions of this form. It follows from corollary 6.4.3.4. that there cannot be more coverings of CCC_r on CCC_s. □

6.4.4. Coverings of the 4-pin shuffle. Recall the definition of the 4-pin shuffle from section 1.6.3. and the definition of step-simulation from section 5.3.1.

Lemma 6.4.4.1. Every covering of S_n on S_{n-k} is a uniform step-simulation.

Definition. Let $f: S_n \rightarrow S_{n-k}$ be a step-simulation. $\tilde{f}: S_{k+1} \rightarrow S_1$ is defined by $\tilde{f}(b_1...b_{k+1}) = f_1(b_1...b_{k+1}o..o)$.

The following result is very similar to theorem 5.3.3.2. and can be obtained in a similar way.

Theorem 6.4.4.2. The mapping π, defined by $\pi(f)=\tilde{f}$ from the step-simulations $S_n \rightarrow S_{n-k}$ to the step-simulations $S_{k+1} \rightarrow S_1$ is a bijection, with the following further properties:

1. If f is uniform, then $\pi(f)=\tilde{f}$ is uniform
2. $f(b_1...b_n)=\pi(f)(b_1...b_{k+1}).\pi(f)(b_2...b_{k+2})...\pi(f)(b_{n-k}...b_n)$.

Lemma 6.4.4.3. Let f cover S_n on S_{n-k}. Then:

(i) $\{\tilde{f}(o\ b_1..b_k),\ \tilde{f}(1\ b_1..b_k)\} = \{o,1\}$, for all $b_1..b_k \in (\frac{o}{1})^k$.

(ii) $\{\tilde{f}(b_1..b_k\ o),\ \tilde{f}(b_1..b_k\ 1)\} = \{o,1\}$, for all $b_1..b_k \in (\frac{o}{1})^k$.

Proof.

(i) $b_1..b_k o..o$ has predecessors $ob_1..b_k o..o$ and $1b_1..b_k o..o$. Hence $f(ob_1..b_k o..o)$ and $f(1b_1..b_k o..o)$ must be different: they can only be different in their first coordinate, because they are both predecessors of $f(b_1..b_k o..o)$, so $\tilde{f}(ob_1..b_k) \neq \tilde{f}(1b_1..b_k)$.

(ii) is proved by similar argument. □

So, for coverings f of S_n on S_{n-k} we have that for all $x\in(\frac{o}{1})^{k-2}$ $\tilde{f}(oxo) = \tilde{f}(1x1) \neq \tilde{f}(ox1) = \tilde{f}(1xo)$. However, this is also a sufficient condition for a step-simulation f to be a covering.

Lemma 6.4.4.4. Let $f: S_n \rightarrow S_{n-k}$ be step-simulating, and let for all $x \in (\frac{o}{1})^{k-2}$ $\tilde{f}(oxo) = \tilde{f}(1x1) \neq \tilde{f}(1xo) = \tilde{f}(1xo)$. Then f is a covering.

Proof.

Without much difficulty it can be checked that the conditions for the coverings of directed graphs are fulfilled. □

So we have a necessary and sufficient condition for a step-simulation $S_n \rightarrow S_{n-k}$ to be a covering. With the help of this condition we can completely classify the coverings of S_n on S_{n-k}.

Theorem 6.4.4.5.

a. If k=1 then the coverings of S_n on $S_{n-k} = S_{n-1}$ are the following:

   $f(b_1..b_n) = c_1..c_{n-1}$ with $c_i = (b_i \equiv b_{i+1})$ $(1 \leq i \leq n-1)$

   $\bar{f}(b_1..b_n) = \bar{c}_1..\bar{c}_{n-1}$ with $c_i = (b_i \equiv b_{i+1})$ $(1 \leq i \leq n-1)$

b. If $2 \leq k < n$, then every covering f of S_n on S_{n-k} can be found by choosing for each string $x \in (\frac{o}{1})^{k-1}$ whether

   - $\tilde{f}(oxo) = \tilde{f}(1x1) = o$ and $\tilde{f}(ox1) = \tilde{f}(1xo) = 1$

   or - $\tilde{f}(oxo) = \tilde{f}(1x1) = 1$ and $\tilde{f}(ox1) = \tilde{f}(1xo) = o$

c. For every $n > k \geq 1$ there are exactly $2^{2^{k-1}}$ coverings of S_n on S_{n-k}.

(In section 5.3.3. it was proven that the functions f: b→ b and f: b→$\bar{b}$ are the only possible graph isomorphisms of S_n (= coverings of S_n on S_n).)

6.4.5. Coverings of the shuffle-exchange network. Recall the definition of the shuffle-exchange network from section 1.6.3.

Theorem 6.4.5.1. There do not exists coverings of SE_n on SE_k with $k \nmid n$ and $k \geq 3$.

Proof.

Suppose $k \nmid n$, $k \geq 3$ and f is a covering of SE_n on SE_k. With $R^l(b)$ we denote the string $b_{l+1 (\mathrm{mod}\ n)} \ldots b_n b_1 \ldots b_{l (\mathrm{mod}\ n)}$, i.e. b rotated l positions to the left.

Let $b \in f^{-1}(o^{k-1}1)$.

Lemma 6.4.5.1.1. $f(R^l(b)) = R^l(f(b)) = R^l(o^{k-1}1)$, for all $l \geq o$.

Proof.

With induction. For $l=o$ the lemma is trivially true. Suppose the lemma holds for certain l. Then it holds also for $l+1$:

$f(R^l(b)|_{n-1}\ (R^l(b))_n)$ must be mutually adjacent to $f(R^l(b))$, so must be connected with $f(R^l(b))$ via the exchange edge (we use that $f(R^l(b))$ cannot be of the form $(o1)^{n/2}$ or $(1o)^{n/2}$ (induction hypotheses)). This shows that $f(R^{l+1}(b))$ must be connected with $f(R^l(b))$ via a shuffle-edge, hence $f(R^{l+1}(b)) = R^l(f(R^l(b)) = R^{l+1}(f(b))$. □

In particular we now have: $o^{k-1}1 = f(b) = f(R^n(b)) = R^n(o^{k-1}1)$. So $k|n$. Contradiction. □

Theorem 6.4.5.2. Let $k\geq 3$, $k|n$. The coverings of SE_n on SE_k are given by the following list:

1. f', defined by $f_i'(b_1 \ldots b_n) = \left(\sum_{j=0}^{n/k-1} b_{j.k+i}\right) \bmod 2$ $(1 \leq i \leq k)$.

2. f', defined by $f_i'(b_1 \ldots b_n) = \left(\sum_{j=0}^{n/k-1} b_{j.k+i}\right) \bmod 2$ $(1 \leq i \leq k)$.

Proof.

Because o^n in SE_n has a self-loop, and o^k, 1^k are the only nodes in SE_k with a self-loop one has for every covering f of SE_n on SE_k: $f(o^n) = o^k$ or $f(o^n) = 1^k$. We suppose we have a covering f of SE_n on SE_k with $f(o^n) = o^k$, and will show that f is then necessarily of the form f'. If $f(o^n) = 1^k$ then f is of the form $\overline{f'}$.

Lemma 6.4.5.2.1. If $f(b) = f'(b)$ and $f(b) \notin \{(o1)^{n/2}, (1o)^{n/2}\}$,

then 1. $f(b_1 \ldots b_{n-1}\ \overline{b_n}) = f'(b_1 \ldots b_{n-1}\ \overline{b_n})$.

2. $f(b_2 \ldots b_n\ b_1) = f'(b_2 \ldots b_n\ b_1)$.

3. $f(b_n\ b_1 \ldots b_{n-1}) = f'(b_n\ b_1 \ldots b_{n-1})$.

Proof.

First note $f(b_1 \ldots b_{n-1}\ \overline{b_n})$ must be mutually adjacent to f(b). Hence $f(b_1 \ldots b_{n-1}\ \overline{b_n}) = f'(b_1 \ldots b_{n-1}\ \overline{b_n})$. (Use the definition of f'.) Further use that $f(b_2 \ldots b_n\ b_1)$ must be a successor of f(b), unequal to $f(b_1 \ldots b_n\ \overline{b_n})$. This shows $f(b_2 \ldots b_n\ b_1) = f'(b_2 \ldots b_n b_1)$. Similarly $f(b_n\ b_1 \ldots b_{n-1}) = f'(b_n\ b_1 \ldots b_{n-1})$. □

Without proof we mention the following result:

Lemma 6.4.5.2.2. For every $b \in (\frac{o}{1})^n$ there is a path from o^n to b, such that for every node c on the path, except b: $f'(c) \notin \{(o1)^{n/2}, (1o)^{n/2}\}$.

(If n is odd, it will of course never be the case that $f'(c) \in \{(o1)^{n/2}, (1o)^{n/2}\}$.)

With induction to the length of the path, mentioned in 6.4.5.2.2. one now can prove that f(b) = f'(b) for every $b \in (\frac{o}{1})^n$. □

6.5. Emulations of processor networks with buses.

6.5.1. Introduction. The given concept of emulation can be used for networks that use links for communication (only). However, a realistic processor network can also use buses for communication. Theoretically, a bus connects an arbitrary set of processors. An advantage of the use of buses is that one can keep the diameter of the network (the maximum number of hops needed to go from one processor to another processor) low, without having a large number of connections per processor. In [Wi81] one can find a comparison between a number of different interconnection schemes, including three networks with buses: the "global bus" (all processors are connected to one bus), the "spanning bus hypercube" and the "dual bus hypercube" (for definitions see section 6.5.4.). In this section we extend the notion of emulation to networks that use buses for communication. In section 6.5.2. we show how to model the interconnection structure of a network with buses by means of hypergraphs. Hypergraphs appear to be a natural way to model networks with buses and allow a formal reasoning about these networks without much difficulty. In section 6.5.3. we extend the notion of emulation to hypergraphs, and define the computation and communication cost of emulations. In section 6.5.4. we give efficient emulations of the spanning bus hypercube and the dual bus hypercube, which are the most common architectures of bus-networks (cf. Wittie [Wi81]). Some final remarks are made in section 6.5.5.

6.5.2. The hypergraph model. The interconnection structure of a network that uses links for communication is usually represented by a (possibly directed) graph. For representing the interconnection structure of networks with buses the graph model seems to be inadequate. Each processor connected to a bus can send messages via the bus, to some subset of the processors that are connected to the bus. There cannot be more than one message simultaneously on the same bus. Therefore it does not seem appropriate to replace all the processors connected to the same bus by a clique, (which would result in a graph with edges between nodes v_1,v_2 iff there is a bus to which v_1 and v_2 are connected). In this way one loses (possibly) essential information about the interconnection structure of the network. For instance the global bus network ("all processors are connected to a single bus") is replaced by a complete graph. However a network with a link between each pair of processors allows much more simultaneous message traffic than a global bus network does.

A natural model for networks with buses is provided by hypergraphs. A hypergraph is a 2-tuple (V,E), with V a set of nodes, and E a collection of subsets of V (edges), with

(i) $e \in E \Rightarrow e \neq \emptyset$

(ii) $\bigcup_{e \in E} e = V$.

For the theory of hypergraphs, see e.g. [Bg76]. The processors of the network correspond to the nodes of the hypergraph, the (links and) buses to the edges. We denote edges by a,b,c,d,e, and nodes by v,w,x,y,z. If processor v is connected to bus b, then $v \in b$, in the hypergraph. Note that the hypergraph is not necessarily simple: multiple busses can connect the same set of nodes. For an example of a network with buses and its hypergraph, see fig. 6.5.2.1. and fig. 6.5.2.2.

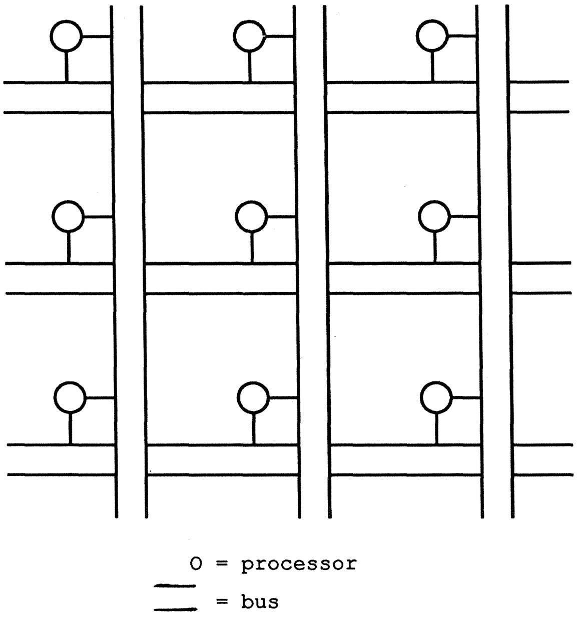


fig. 6.5.2.1.

A processor network with 9 processors and 6 buses

It is possible that processor networks use a combination of links and buses. Links can be full-duplex (the two processors can send a message to each other simultaneously), half-duplex (at most one processor can send at a time) or simplex (messages can be send only in one direction). Half-duplex links can be treated as buses with two connected processors. To incorporate the other two types of links in our model, we define "directed hypergraphs". A directed hypergraph is a 2-tuple (V,E), with V a set of nodes and E a collection of ordered pairs of subsets of V (edges), with

(i) $(d,e) \in E \Rightarrow (d \neq \emptyset \wedge e \neq \emptyset)$

(ii) $\cup \{e \mid \exists d \ (d,e) \in E\} = V$

(iii) $\cup \{d \mid \exists e \ (d,e) \in E\} = V$.

Each node in V represents a processor, each edge (d,e) in E a

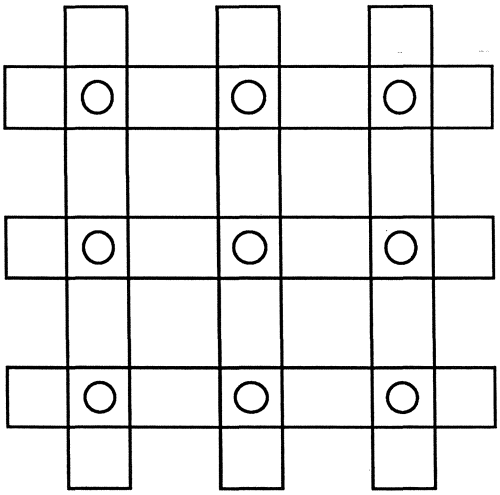


fig. 6.5.2.2. The corresponding hypergraph with 9 nodes and 6 edges

communication medium. The nodes in d represent processors that can send messages via the medium; the nodes in e represent processors that can receive messages via the medium. For a bus (d,e) one has d=e. Full-duplex links are replaced by two simplex links.

6.5.3. Emulations of hypergraphs. Unlike in the case of graph emulations, a mapping of the nodes V_G of a hypergraph $G=(V_G,E_G)$ on the nodes V_H of a hypergraph $H=(V_H,E_H)$ does not immediately induce a mapping of E_G on $V_H \cup E_H$. Therefore we include the mapping of E_G on $V_H \cup E_H$ explicitly in the definition of an emulation.

Definition. Let $G=(V_G,E_G)$, $H=(V_H,E_H)$ be hypergraphs. A mapping $f\colon V_G \cup E_G \rightarrow V_H \cup E_H$ is an emulation function (in short: an emulation) of G on H, if and only if

(i) $f(V_G) \subseteq V_H$, and

(ii) For all $v \in V_G$, $e \in E_G$, $v \in e$:

$$f(e) \in V_H \Rightarrow f(v)=f(e)$$

$$\text{and } f(e) \in E_H \Rightarrow f(v) \in f(e).$$

Clearly, emulation is transitive. (If f emulates G on H, and g emulates H on K, then $g \circ f$ emulates G on K.) For (undirected) graphs (seen as hypergraphs with each edge cardinality 2) the new definition of emulation coincides with the old definition. Let f be an emulation of graph $G=(V_G,E_G)$ on graph $H=(V_H,E_H)$ (by the new definition), and suppose $(v_1,v_2) \in E_G$. If $f((v_1,v_2)) \in V_H$, then $f(v_1) \in f((v_1,v_2))$ and $f(v_2) \in f((v_1,v_2))$, so $f(v_1)=f(v_2)$ or $(f(v_1),f(v_2)) \in E_H$.

Let f be an emulation of hypergraph $G=(V_G,E_G)$ on hypergraph $H=(V_H,E_H)$. Any processor $w \in V_H$ must actively emulate the processors in $f^{-1}(w) \cap V_G$. When $v \in f^{-1}(w) \cap V_G$ communicates information via a bus b to neighboring processors $x_1,\ldots,x_m \in b$, them w must either communicate the corresponding information internally (iff $w=f(b) \in V_H$), or communicate the information via bus f(b) to the processors $f(x_1),\ldots,f(x_m)$,(iff $f(b) \in E_H$).

To determine the factor by which the emulation "slows down" a computation on the network we distinguish between the time needed for computations inside the processors, and the time needed for communication between the processors. If all processors act synchronously in G, then the time needed for computations will be increased by a factor, proportional to $\max\limits_{w \in V_H} |f^{-1}(w) \cap V_G|$.

Definition. Let f emulate hypergraph $G=(V_G,E_G)$ on hypergraph $H=(V_H,E_H)$. The computation cost of f is $cc(f)= \max\limits_{w \in V_H} |f^{-1}(w) \cap V_G|$.

Definition. Let f be as above. f is computationally uniform, iff $cc(f)=|V_G|/|V_H|$.

It means that for a computationally uniform emulation f, for every processor $w \in V_H$, $|f^{-1}(w) \cap V_G| = cc(f)$: every processor w in H emulates the same number of processors in G.

The factor by which the communication time is slowed down can be estimated by $\max_{e \in E_H} |f^{-1}(e)|$ (=the maximum number of buses a bus has to emulate.) This factor can be 0, in the degenerate case that all processors that can communicate with each other are mapped on the same node. In general all processors can communicate with each other using one or more communication steps (i.e. the hypergraph is "connected"), so if for an emulation f of a connected network G on a network H the maximum is 0, then f maps every node of G on the same node of H. In general, there will be at least one $e \in E_H$, with $f^{-1}(e) \neq \emptyset$. The factor $\max_{e \in E_H} |f^{-1}(e)|$ is obtained if we let each emulated bus b in E_G use the bus e=f(b) once every $\max_{e \in E_H} |f^{-1}(e)|$ timesteps. (The estimation is not necessarily optimal, but in general it is.)

<u>Definition</u>. Let f emulate hypergraph $G=(V_G,E_G)$ on hypergraph $H=(V_H,E_H)$. The communication cost of f is $comc(f) = \max_{e \in E_H} |f^{-1}(e)|$.

<u>Definition</u>. Let f be as above. f is communicationally uniform, iff for all $e \in E_H$ $|f^{-1}(e)| = comc(f)$.

Again, computationally uniform, and communicationally uniform emulations are transitive as relations. For directed hypergraphs, the definition of emulation becomes as follows:

<u>Definition</u>. Let $G=(V_G,E_G)$ and $H=(V_H,E_H)$ be directed hypergraphs. $f : V_G \cup E_G \rightarrow V_H \cup E_H$ is an emulation function (in short: an emulation) of G on H, iff

(i) $f(V_G) \subseteq V_H$

(ii) For all $v,w \in V_H$ $e=(a,b) \in E_G$, $v \in a$, $w \in b$:

$f(e) \in V_H \Rightarrow f(v) = f(w) = f(e)$

and $f(e) \in E_H \Rightarrow$ one can write $f(e) = (c,d)$, with $f(a) \subseteq c$, $f(b) \subseteq d$.

The definitions of computation and communication cost, and of computationally and communicationally uniform are the same for hypergraphs and directed hypergraphs.

For graphs (seen as hypergraphs with each edge of cardinality 2) and directed graphs (seen as directed hypergraphs, with each edge consisting of an ordered pair of sets, each with one element) the definitions of "computationally uniform" coincide with the old definition of "(computationally) uniform" (cf. section 5.1.).

6.5.4. Emulations of common networks with buses.

6.5.4.1. Emulations of the spanning bus hypercube. The d-dimensional spanning bus hypercube with width n has nodes, that can be seen as if they lie in a n×n×...×n integer grid with dimension d. That is, each node can be specified with d integer coordinates in the range $\{0,...,n-1\}$. Each node is connected to d buses, each bus going in a different dimension, connecting all the nodes whose coordinates agree in all other dimensions.

Definition. The d-dimensional spanning bus hypercube with width n is the hypergraph $SB^{d,n} = (V^{d,n}, E^{d,n})$, with $V^{d,n} = \{(x_1,x_2,...,x_d) \mid \forall i, 1 \le i \le d: 0 \le x_i \le n-1\}$ and $E^{d,n} = \{e_{x_1...x_{i-1}x_{i+1}...x_d,i} \mid x_1,...,x_{i-1},x_{i+1},...,x_d \in \{0,...,n-1\}, 1 \le i \le d\}$, with $e_{x_1...x_{i-1}x_{i+1}...x_d,i} = \{ (x_1,...,x_{i-1},x_i,x_{i+1},...,x_d) \mid 0 \le x_i \le n \}$.

An example with d=3 and n=4 is given in fig. 6.5.4.1.

The one-dimensional spanning bus hypercubes are the global bus networks. The spanning bus hypercubes with width 2 are very similar to the cube networks. For an analysis of the emulations of the cube networks see section 5.5.

One can emulate the d-dimensional spanning bus hypercube on the d-k-dimensional spanning bus hypercube with the same width for any $1 \le k \le d-1$ by a straightforward projection.

Theorem 6.5.4.1.1. The function $f: V^{d,n} \cup E^{d,n} \rightarrow V^{d-k,n} \cup E^{d-k,n}$, given by

- $f((x_1,\ldots,x_d)) = (x_1,\ldots,x_{d-k})$ and
- $f(e_{x_1 \ldots x_{i-1} x_{i+1} \ldots x_d, i}) = e_{x_1 \ldots x_{i-1} x_{i+1} \ldots x_{d-k}, i}$ iff $i \le d-k$ and
- $f(e_{x_1 \ldots x_{i-1} x_{i+1} \ldots x_d, i}) = (x_1,\ldots,x_{d-k})$ iff $i > d-k$

is a computationally and communicationally uniform emulation of $SB^{d,n}$ on $SB^{d-k,n}$ $(d,k,d-k,n \in N^+)$, with $cc(f) = comc(f) = n^k$.

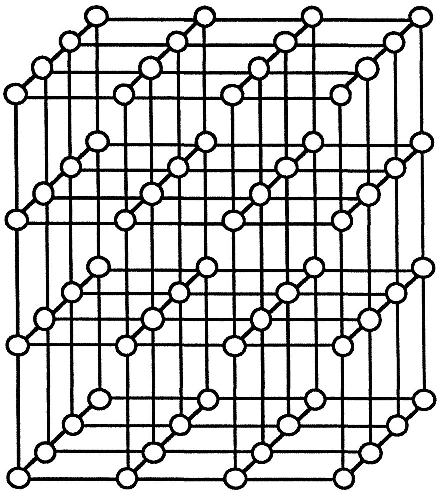


fig. 6.5.4.1.

3-dimensional spanning bus hypercube with width 4.

Proof.

It is clear that $f(V^{d,n}) \subseteq V^{d-k,n}$. Consider node $x=(x_1,\dots,x_d) \in V^{d,n}$ and edge $e=e_{x_1\dots x_{i-1}x_{i+1}\dots x_d,i} \in E^{d,n}$. If $i\le d-k$ then $f(e) \in E^{d-k,n}$, and $f(x)=(x_1,\dots,x_{d-k}) \in e_{x_1\dots x_{i-1}x_{i+1}\dots x_{d-k},i}=f(e)$. If $i>d-k$, then $f(e) \in V^{d-k,n}$ and $f(x)= (x_1,\dots,x_{d-k})= f(e)$. So f is an emulation.
For all y $(y_1,\dots,y_{d-k}) \in V^{d-k,n}$, $|f^{-1}(y) \cup V^{d,n}| = |\{(y_1,\dots,y_{d-k},y_{d-k+1},\dots,y_d) \mid y_{d-k+1},\dots,y_d \in \{0,\dots,n-1\}\}| = n^k$. So f is computationally uniform with $cc(f) = n^k$.
For all $e = e_{y_1\dots y_{i-1}y_{i+1}\dots y_{d-k},i} \in E^{d-k,n}$, $|f^{-1}(e)| = |\{e_{y_1\dots y_{i-1}y_{i+1}\dots y_d,i} \mid y_{d-k+1},\dots,y_d\in\{0,..,n-1\}\}| = n^k$. So f is communicationally uniform with $comc(f) = n^k$. □

For the next result, let c be any integer with $c|n$.

Theorem 6.5.4.1.2. The function $f:V^{d,n} \cup E^{d,n} \to V^{d,\frac{n}{c}} \cup E^{d,\frac{n}{c}}$, given by
- $f((x_1,\dots,x_d)) = (x_1\mathrm{mod}\frac{n}{c},x_2\mathrm{mod}\frac{n}{c},\dots,x_d\mathrm{mod}\frac{n}{c})$ and
- $f(e_{x_1\dots x_{i-1}x_{i+1}\dots x_d,i}) =$
$$e_{(x_1\mathrm{mod}\frac{n}{c})\dots(x_{i-1}\mathrm{mod}\frac{n}{c})(x_{i+1}\mathrm{mod}\frac{n}{c})\dots(x_d\mathrm{mod}\frac{n}{c}),i}$$
is a computationally and communicationally uniform emulation of $SB^{d,n}$ on $SB^{d,n/c}$ $(d,n,c \in N^+,\ c|n)$, with $cc(f)=c^d$; $comc(f)=c^{d-1}$.

Proof.

It is clear that $f(V^{d,n}) \subseteq V^{d,\frac{n}{c}}$. Consider $x=(x_1,\dots,x_d) \in V^{d,n}$ and $e=e_{x_1\dots x_{i-1}x_{i+1}\dots x_d,i} \in E^{d,n}$. $f(e) \in E^{d,\frac{n}{c}}$ and $f(x) = (x_1\mathrm{mod}\frac{n}{c},\dots,x_d\mathrm{mod}\frac{n}{c}) \in e_{(x_1\mathrm{mod}\frac{n}{c})\dots(x_{i-1}\mathrm{mod}\frac{n}{c})(x_{i+1}\mathrm{mod}\frac{n}{c})\dots(x_d\mathrm{mod}\frac{n}{c}),i} = f(e)$, hence f is an emulation.
For all $y = (y_1,\dots,y_d) \in V^{d,n/c}$, $|f^{-1}(y) \cap V^{d,n}| = |\{(x_1,\dots,x_d) | \forall_i, 1\le i\le d\ \ x_i\mathrm{mod}\frac{n}{c} = y_i\}| = c^d$. So f is computationally uniform with $cc(f)=c^d$.
For all $e= e_{y_1\dots y_{i-1}y_{i+1}\dots y_d,i}$, $|f^{-1}(e)|= |\{e_{x_1\dots x_{i-1}x_{i+1}\dots x_d,i}$ |

$\forall j,\ 1 \le j \le d,\ j \ne i;\ x_j \bmod \frac{n}{c} = y_j\}| = c^{d-1}$. So f is communicationally uniform with $comc(f) = c^{d-1}$. □

6.5.4.2. Emulations of the dual bus hypercube. Wittie [Wi81] proposed a variant of the spanning bus hypercube, which uses significantly fewer connections per node: the dual bus hypercube. Nodes in the dual bus hypercube can again be seen as points in a d-dimensional integer grid of size $n \times n \times \ldots \times n$, with $n+1 \ge d \ge 2$. In the dual bus hypercube each node is connected to two buses. In one dimension, say the first, buses connect nodes, as in the spanning bus hypercube. (These are n^{d-1} buses, each connected to n nodes.) Look at the d-1 dimensional layers, that are perpendicular on these buses. In the first layer, buses go in the second dimension, each connecting n nodes that agree in all but the second coordinate. In the second layer, buses go in the third dimension, etc., so in the j'th layer, buses go in the $2+(j-1)\bmod(d-1)$'th dimension. In each layer there are n^{d-2} buses, each connected to n nodes. The advantage of a dual-bus hypercube over the spanning bus hypercube is that each node has only two connections to a bus. The diameter of the network however is 2d-1, only a factor of about 2 larger than the diameter of the spanning bus hypercube (=d).

Definition. The d-dimensional dual bus hypercube with width n is the hypergraph $DB^{d,n} = (V^{d,n}, \tilde{E}^{d,n})$, with $V^{d,n} = \{(x_1,\ldots,x_d) \mid \forall\ i,\ 1 \le i \le d: 0 \le x_i \le n-1\}$ and $\tilde{E}^{d,n} = \{e_{x_1 \ldots x_{i-1} x_{i+1} \ldots x_d, i} \mid x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_d \in \{0,\ldots,n-1\}$ and $(i=1$ or $(2 \le i \le d \wedge i = 2+(x_1-1)\bmod(d-1)))\}$, with $e_{x_1 \ldots x_{i-1} x_{i+1} \ldots x_d, i} = \{(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_d) \mid 0 \le x_i \le n-1\}$.

For d=1 and d=2 the d-dimensional dual bus hypercube $DB^{d,n}$ and the d-dimensional spanning bus hypercube are the same. One has to take $n+1 \ge d$, to let the dual bus hypercube be connected.

An example of the dual bus hypercube, with d=3 and w=4 is given in fig. 6.5.4.2.

For the projections of the dual bus hypercubes on lower dimensional dual

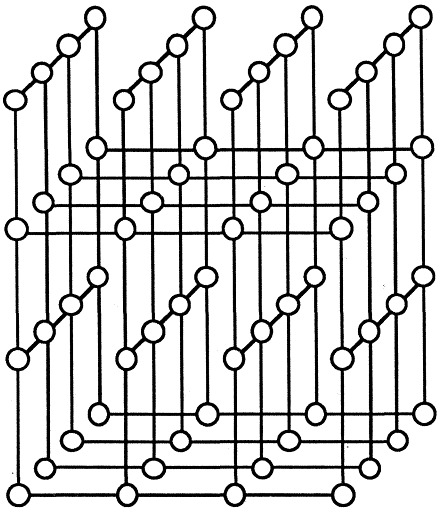


fig.6.5.4.2.
3 dimensional dual bus hypercube with width 4.

bus hypercubes with the same width we need the following result:

Lemma 6.5.4.2.1. Let $d,d',n \in N^+$, and $d>d'\geq 2$. There is a bijection $\psi^{d,d',n}$ from $\{0,\ldots,n-1\}$ to $\{0,\ldots,n-1\}$, such that if $(x-1)\bmod(d-1)+2 \leq d'$ then $(x-1)\bmod(d-1)+2 = (\psi^{d,d',n}(x)-1)\bmod(d'-1)+2$.

Proof.

For each i in $\{2,\ldots,d'\}$, one has that $|\{x|0\leq x\leq n-1$ and $(x-1)\bmod(d-1)+2 = i\}| \leq |\{x|0\leq x\leq n-1$ and $(x-1)\bmod(d'-1)+2 = i\}|$. So we can map all x with $(x-1)\bmod(d-1)+2 \leq d'$ in a one-to-one manner, such that $(x-1)\bmod(d-1)+2 = (\psi^{d,d',n}(x)-1)\bmod(d'-1)+2$, and map the x with $(x-1)\bmod(d'-1)+2 > d'$ onto the remaining nodes. □

Theorem 6.5.4.2.2. Let d,d',n and $\psi=\psi^{d,d',n}$ be as above. The function $f:V^{d,n} \cup \tilde{E}^{d,w} \rightarrow V^{d',n} \cup \tilde{E}^{d',n}$, given by

- $f((x_1,\ldots,x_d))= (\psi(x_1),x_2,\ldots,x_{d'})$ and

- $f(e_{x_1...x_{i-1}x_{i+1}...x_{d'},i}) = e_{x_2...x_{d'},1}$, iff $i=1$ and
- $f(e_{x_1...x_{i-1}x_{i+1}...x_{d'},i}) = e_{\psi(x_1)x_2...x_{i-1}x_{i+1}...x_{d'},i}$, iff $2 \le i \le d'$ and
- $f(e_{x_1...x_{i-1}x_{i+1}...x_{d'},i}) = (\psi(x_1),x_2,...,x_{d'})$, iff $i>d'$,

is a computationally and communicationally uniform emulation of $DB^{d,n}$ on $DB^{d',n}$, with $cc(f) = comc(f) = n^{d-d'}$.

Proof.

First we have to show that f is a correct mapping, i.e. we have to show that for $2 \le i \le d'$, if $e_{x_1...x_{i-1}x_{i+1}...x_{d'},i} \in \tilde{E}^{d,n}$, then $e_{\psi(x_1)x_2...x_{i-1}x_{i+1}...x_{d'},i} \in \tilde{E}^{d',n}$. This is because $e_{x_1...x_{i-1}x_{i+1}...x_{d'},i} \in \tilde{E}^{d,n} \Rightarrow i=2+(x_1-1)\bmod(d-1) \Rightarrow$ (use that $i \le d'$) $2+(x_1-1)\bmod(d-1) = 2+(\psi(x_1)-1)\bmod(d'-1)=i \Rightarrow e_{\psi(x_1)x_2...x_{i-1}x_{i+1}...x_{d'},i} \in \tilde{E}^{d',n}$.

Now we show f is an emulation. It is obvious that $f(V^{d,n}) \subseteq V^{d',n}$. Consider $x=(x_1,...,x_d) \in V^{d,n}$ and edge $e=e_{x_1...x_{i-1}x_{i+1}...x_{d'},i} \in \tilde{E}^{d,n}$. If $i=1$, then $f(x) = (\psi(x_1),x_2,...,x_{d'}) \in e_{x_2...x_{d'},1}=f(e)$. If $2 \le i \le d'$, then $f(x) = (\psi(x_1),x_2,...,x_{d'}) \in e_{\psi(x_1)x_2...x_{i-1}x_{i+1}...x_{d'},i} = f(e)$. If $i>d$, then $f(x) = (\psi(x_1),x_2,...,x_{d'}) = f(e)$. Hence f is an emulation.

For all $y=(y_1,...,y_{d'}) \in V^{d',n}$, $|f^{-1}(x) \cap V^{d,n}| = |\{(\psi^{-1}(y_1),y_2,...,y_{d'},y_{d'+1},...,y_d) \mid y_{d'+1},...,y_d \in \{0,..,n-1\}\}| = n^{d-d'}$. (We use that ψ is bijective.) Hence f is computationally uniform and $cc(f) = n^{d-d'}$.

For all $e=e_{y_2...y_{d'},1}$, $|f^{-1}(e)| = |\{e_{y_2...y_{d'},1} | y_{d'+1},...,y_d \in \{0,..,n-1\}\}| = n^{d-d'}$. For all $e=e_{y_1...y_{i-1}y_{i+1}...y_{d'},i}$, with $2 \le i \le d'$, $|f^{-1}(e)| = |\{e_{\psi^{-1}(y_1)y_2...y_{i-1}y_{i+1}...y_{d'}y_{d'+1}...y_{d'},i} \mid y_{d'+1},...,y_d \in \{0,...,n-1\}\}| = n^{d-d'}$. Hence f is communicationally uniform, and $comc(f) = n^{d-d'}$. □

Theorem 6.5.4.2.3. Let $d,n,c \in N^+$, $c|n$, $(d-1)|(\frac{n}{c})$.

The function $f: V^{d,n} \cup \tilde{E}^{d,n} \to V^{d,\frac{n}{c}} \cup \tilde{E}^{d,\frac{n}{c}}$, given by

- $f((x_1,\ldots,x_d)) = (x_1 \bmod \frac{n}{c},\ldots,x_d \bmod \frac{n}{c})$ and
- $f(e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i}) =$

$$e_{(x_1 \bmod \frac{n}{c})\ldots(x_{i-1} \bmod \frac{n}{c})(x_{i+1} \bmod \frac{n}{c})\ldots(x_d \bmod \frac{n}{c}),i}$$

is a computationally and communicationally uniform emulation of $DB^{d,n}$ on $DB^{d,\frac{n}{c}}$. $cc(f)=c^d$; $comc(f)=c^{d-1}$.

Proof.

It is obvious that $f(V^{d,n}) \subseteq V^{d,\frac{n}{c}}$. f is correct mapping: $e=e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i} \in \tilde{E}^{d,n} \Rightarrow ((i=1) \text{ or } (2\le i\le d \wedge i= (x_1-1)\bmod(d-1))+2)$ $\Rightarrow (i=1 \text{ or } (2\le i\le d \wedge i= (x_1 \bmod(\frac{n}{c})-1)\bmod(d-1)+2)$ (use that $(d-1)\mid(\frac{n}{c})$) $\Rightarrow$ $f(e) \in \in\tilde{E}^{d,\frac{n}{c}}$. The remainder of the proof is similar to the proof of theorem 6.5.4.1.2. □

We next consider emulations of $DB^{d,n}$ on $DB^{d,n/c}$, with $(d-1) \nmid \frac{n}{c}$. The next result shows that there exist efficient, non-uniform emulations of $DB^{d,n}$ on $DB^{d,n/c}$, if $(d-1) \nmid \frac{n}{c}$: we have an extra factor of at most 2 in the formulas for the computation and communication cost, compared with the costs of the uniform emulations of $DB^{d,n}$ on $DB^{d,n/c}$, with $(d-1)\mid\frac{n}{c}$.

Lemma 6.5.4.2.4. Let $d,n,n' \in N^+$, $n'\mid n$, $n'\ge d-1$. Let $c=n/n'$. There exists a mapping ψ: $\{0,\ldots,n-1\} \to \{0,\ldots,n'-1\}$, such that

(i) $\forall x$, $0\le x\le n-1$: $x \bmod(d-1)= \psi(x)\bmod(d-1)$

(ii) $\forall y$, $0\le y\le n'-1$: $|\psi^{-1}(y)| \le \lceil n/((d-1)\cdot\lfloor\frac{n}{d-1}\rfloor)\rceil \le 2c$.

Proof.

Write $n'=k(d-1)+l$, with $0\le l<d-1$. The mapping ψ: $\{0,\ldots,n-1\} \to \{0,\ldots,n'-1\}$, given by $\psi(x)=x \bmod(k(d-1))$ fulfills property (i) and (ii): $\forall$ x, $0\le x\le n-1$: $x \bmod(d-1) = (x \bmod(k(d-1)))\bmod(d-1)$; and $\forall y$, $0\le y\le n'-1$, $|\psi^{-1}(y)|=0$, if $y\ge k(d-1)$, and $|\psi^{-1}(y)| = |\{x\mid 0\le x\le n-1, x \bmod(k(d-1))=y\}| = \lceil n/(k(d-1))\rceil$, if $y<k(d-1)$. Note that $k=\lfloor\frac{n'}{d-1}\rfloor$, and $\lceil n/(k(d-1))\rceil = \lceil (ck(d-1)+cl)/(k(d-1))\rceil \le 2c$. □

Theorem 6.5.4.2.5. Let d,n,n',ψ,c be as above. The function $f\colon V^{d,n} \cup \tilde{E}^{d,n} \rightarrow V^{d,n} \cup \tilde{E}^{d,n'}$, given by

- $f((x_1,\ldots,x_d))=(\psi(x_1),x_2\bmod n',x_3\bmod n',\ldots,x_d\bmod n')$ and
- $f(e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i}) = e_{x_2\bmod n'\ldots x_d\bmod n',i}$, iff $i=1$, and
- $f(e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i})=$

  $e_{\psi(x_1)x_2\bmod n'\ldots x_{i-1}\bmod n' x_{i+1}\bmod n'\ldots x_d\bmod n',i}$, iff $2\le i\le d$

is an emulation of $DB^{d,n}$ on $DB^{d,n'}$, with $cc(f) \le \lceil n/((d-1)\cdot\lfloor \frac{n'}{d-1}\rfloor)\rceil\cdot c^{d-1} \le 2c^d$, and $comc(f) \le \lceil n/((d-1)\cdot\lfloor \frac{n'}{d-1}\rfloor)\rceil\cdot c^{d-2} \le 2c^{d-1}$.

Proof.

We first show that f is a correct mapping. For $e=e_{x_2\ldots x_d,1}$, it is clear that $f(e) \in \tilde{E}^{d,n}$. Now let $2\le i\le d$. $e=e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i} \in \tilde{E}^{d,n} \Rightarrow$ $i=(x_1-1)\bmod(d-1)+2 \Rightarrow i=(\psi(x_1)-1)\bmod(d-1)+2 \Rightarrow$ $f(e)=e_{\psi(x_1)x_2\bmod n'\ldots x_{i-1}\bmod n' x_{i+1}\bmod n'\ldots x_d\bmod n',i} \in\tilde{E}^{d,n'}$. So f is a correct mapping. f is an emulation of $DB^{d,n}$ on $DB^{d,n'}$: clearly $f(V^{d,n}) \subseteq V^{d,n'}$ and it is easy to check, that for all $x \in V^{d,n}$, $e \in \tilde{E}^{d,n}$, $x \in e \Rightarrow f(x) \in f(e)$.

Finally observe that for all $y=(y_1,\ldots,y_d) \in V^{d,n'}$: $|f^{-1}(y)|= |\psi^{-1}(y_1)|.c^{d-1}$, and for all $e=e_{x_1\ldots x_{i-1}x_{i+1}\ldots x_d,i} \in \tilde{E}^{d,n'}$: $|f^{-1}(e)|=c^{d-1}$, if $i=1$ and $|f^{-1}(e)| = |\psi^{-1}(y_1)|.c^{d-2}$, if $2\le i\le d$. The estimations of $cc(f)$ and $comc(f)$ now follow from lemma 6.5.4.2.4. □

CHAPTER SEVEN

EMULATIONS: COMPLEXITY RESULTS

7.1. Introduction. In this chapter we proceed with the analysis of the notion of emulations, but now we focus our attention on the complexity of the problem of finding uniform emulations between given networks of certain characteristics (with a sequential algorithm). Consider the following problem:

[UNIFORM EMULATION]

Instance: Connected graphs $G=(V_G,E_G)$, $H=(V_H,E_H)$

Question: Is there a uniform emulation of G on H?

The variant of this problem in which the computation factor $c=|V_G|/|V_H|$ is fixed will be called c-UNIFORM EMULATION. Note that the related question whether G can be emulated on H without the constraint of uniformity always yields the answer yes (provided H contains at least one node): every constant function is an emulation. We assume that the reader is familiar with the theory of NP-completeness (see Garey and Johnson[GJ79] or section 1.7. of this thesis).

We will prove that c-UNIFORM EMULATION is NP-complete, for every $c \in N^+$. The problem remains NP-complete under several additional constraints on the guest and host networks. Realistic constraints are: the graphs are connected (or strongly connected in the case of directed graphs) and the nodes of the graph have a bounded degree. We will mainly consider connected and strongly connected graphs, respectively.

This chapter is organized as follows. In section 7.2. we prove that c-UNIFORM EMULATION is NP-complete and consider graphs of bounded degree. In section 7.3. we consider the problem for the case that the host graph H is required to be of a specific type, for the following types of networks: the two-dimensional grid network, the cube network, the shuffle-exchange network and the 4-pin shuffle. In section 7.4. we consider the problem for the case that the host graph H is required to be a path or a ring network. In section 7.5. we prove that UNIFORM

EMULATION is NP-complete, even if the host graph H is fixed to any connected graph H, that is not complete. In section 7.6. we introduce the notion of computation cost of an emulation, and we comment on the existence of "good", polynomial time approximation algorithms for minimizing the computation cost of an emulation of a network G on a network H. In section 7.7. we consider the complexity of finding coverings of a network G on a network H. In section 7.8. we comment on the complexity of finding uniform emulations between networks with buses.

7.2. The complexity of finding uniform emulations for graphs of bounded degree. In this section we prove c-UNIFORM EMULATION te be NP-complete, for every (fixed) $c \geq 1$ and consider graphs of bounded degree. In section 7.2.1. the results on undirected graphs are presented. In section 7.2.2. we consider directed graphs.

7.2.1. Undirected graphs of bounded degree.

Theorem 7.2.1.1. UNIFORM EMULATION is NP-complete.

Proof. It is easy to see that this problem is in NP. (Guess a function f: $V_G \rightarrow V_H$, and check whether it is an emulation and whether it is uniform.) To prove NP-completeness, observe that one can polynomially transform HAMILTONIAN CIRCUIT to the problem. Let $H=(V_H,E_H)$ be a connected, undirected graph and let $G=(V_G,E_G)$ be the undirected graph consisting of one cycle of $|V_H|$ nodes. G can be uniformly emulated on H if and only if H contains a Hamiltonian circuit. □

Theorem 7.2.1.2. For every c_1, $c_2 \in N^+$, the following problem is NP-complete:

Instance: Connected, undirected graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, with each node of V_G of degree at most c_1+1 and each node of V_H of degree at most c_2+2, and $|V_G|=c_1 \cdot c_2 \cdot |V_H|$.

Question: Is there a uniform emulation of G on H?

Proof. Clearly the problem is in NP. To prove NP-completeness, we transform HAMILTONIAN CIRCUIT for undirected graphs with nodes of degree exactly 3 and no cycles of length 3 or 4, to this problem. This version of HAMILTONIAN CIRCUIT is NP-complete [GJ79,GJT76]. Note that the result in [GJT76] did not state the last constraint, but the (planar) graphs resulting from the construction in [GJT76] indeed do not have cycles of length 3 or 4.

Let an arbitrary graph $H_0=(V_0,E_0)$ be given with nodes of degree 3 and no cycles of length 3 or 4. Construct a graph $H=(V_H,E_H)$ with $|V_H|=|V_0|+|V_0|\cdot 2\cdot(c_2-1)$ nodes by adding to each node of H_0 c_2-1 extra "branches" of 2 nodes. An example of this transformation is given in fig. 7.2.1.1. (with $c_2=3$).

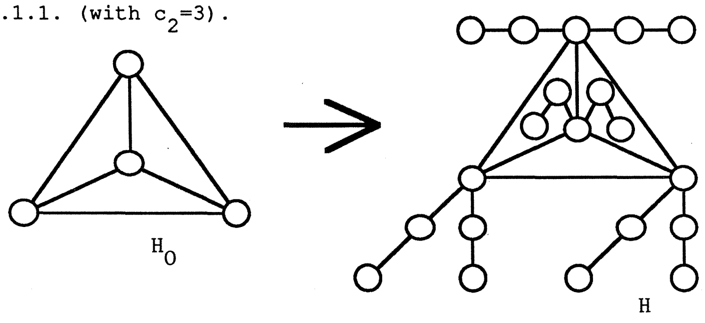


fig. 7.2.1.1.

Let $n=c_2|V_H|=c_2(|V_0|+|V_0|\cdot 2\cdot(c_2-1))$. Let $G=(V_G,E_G)$ be the graph defined by $V_G=\{(i,j)|\ 0\le i\le n-1\ \wedge\ 1\le j\le c_1\}$, and $E_G=\{((i_1,j_1),(i_2,j_2))|(i_1,j_1),\ (i_2,j_2)\in V_G \wedge ((i_1=i_2 \wedge j_1\neq j_2) \vee (j_1=j_2 \wedge i_1=i_2\pm 1(\text{mod } n))))\}$. So G consists of n cliques of c_1 nodes and thus has $c_1\cdot c_2\cdot|V_1|$ nodes; each node is also connected to a node in the "successor clique", and a node in the "predecessor clique". We assume for the sake of argument that G is oriented, say, counter-clockwise. An example of G, with n=6 and $c_1=3$ is given in fig. 7.2.1.2.

The construction of G and H can be done in time polynomial in $|V_0|$, and G and H satisfy the conditions of the problem.

Claim 7.2.1.2.1. H_0 contains a Hamiltonian circuit if and only if there is a uniform emulation of G on H.

Proof. Suppose we have a uniform emulation f of G on H. Let A_j =

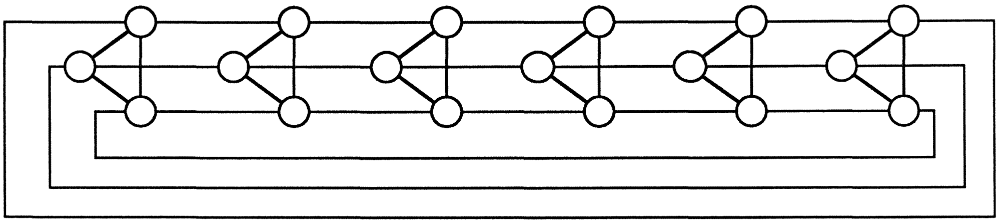


fig. 7.2.1.2.

$\{(i,j)|0\le i\le n-1\} \subseteq V_G$, for every j, $1\le j\le c_1$. Note that for every j, $1\le j\le c_1$, A_j is a cycle of G. We consider two cases:

Case I : $c_2\ge 2$. Let $v \in V_0 \subseteq V_H$ and let j be fixed, $1\le j\le c_1$. Now look at the set of successors of nodes $(i,j) \in G$ with $f((i,j))=v$. For every branch attached to v in H, there is at least one node of V_G that is mapped upon the "end of the branch", (that is, the node with degree 1). This node is connected to a node of the form (x,j), so the "cycle" $f(A_j)$ will visit this branch. So there are c_2-1 successors of nodes (i,j) with $f((i,j))=v$ that are mapped upon branches of v. There must be also one such successor that is mapped upon another node $v' \in V_0 \subseteq V_H$. So there are at least c_2 nodes of A_j that are mapped upon v, for every j, $1\le j\le c_1$. Due to the uniformity of f, this number must be exactly c_2 for all j, $1\le j<=c_1$. (This means essentially that the "cycle" $f(A_j)$ after visiting a node $v \in V_0$ for the first time, must visit successively all branches of v, and then leave v to another node $v' \in V_0$ and cannot return to v for a second time thereafter.) Now we have that for each node $v \in V_0 \subseteq V_H$ there is a unique $v' \in V_0 \subseteq V_H$ such that there is a $i \in \{0,\ldots,n-1\}$ with $f((i,1))=v$ and $f((i+1)\text{mod } n)=v'$. We can call v' the successor of v. By successively visiting the successors of the nodes in $V_0 \subseteq V_H$ we have a Hamiltonian circuit of H_0.

Case II: $c_2=1$. First notice that $H=H_0$. We claim that there are no $i_1\ne i_2$ with $f((i_1,1))=f((i_2,1))$. Suppose there are. In the following we let + be the addition modulo n and − the subtraction modulo n.

Let $J_1 = \{j|f((i_1,j)) = f((i_1,1))\}$, $J_2 = \{j|f((i_1,j) \ne f((i_1,1))\}$, $v_1 = f((i_1,1))$, $j_2 \in J_2$, and $v_2 = f((i_1,j_2))$.

Claim: $f((i_1+1,J_1)) \neq v_1$ and $|f((i_1+1,J_1)| = 1$.

Proof. First suppose $f((i_1+1,\ J_1))=v_1$. The nodes in $f((i_1+1,\ J_2))$ must be equal of adjacent to $f((i_1,J_2)) = v_2$ and to $f((i_1+1,J_1)) = v_1$. As $H=H_0$ does not have 3-cycles, we have $f((i_1+1,J_2)) \subseteq \{v_1,v_2\}$. So $f^{-1}(\{v_1,v_2\}) = \{(i_1,j^*)|1\leq j^*\leq c_1\} \cup \{(i_1+1,j^*)|1\leq j^*\leq c_1\}$. Now $f((i_1-1,1))$ must be adjacent to $v_1 = f((i_1,1))$ (it cannot be equal to it due to the uniformity of f), $f((i_1-1,\ j_2))$ must be adjacent to $v_2=f((i_1,j_2))$ (again it cannot be equal to it) and $f((i_1-1,1))$ and $f((i_1-1,j_2))$ must be equal or adjacent, so $H=H_0$ contains a 3-cycle or a 4-cycle. Contradiction. Suppose $|f((i_1+1,J_1))|\geq 2$. Notice that every node in $f((i_1+1,J_1))$ is equal or adjacent to v_1 and H does not contain 3-cycles, so $v_1 \in f((i_1+1,J_1))$. If $v_2 \in f((i_1+1,J_1))$ then $f^{-1}((v_1,v_2)) = \{(i_1,j^*)|1\leq j^*\leq c_1\} \cup \{(i_1+1,j^*)|1\leq j^*\leq c_1\}$ and a contradiction can be obtained as before. So there is a node $v_3 \notin \{v_1,v_2\}$ with $v_3 \in f((i_1+1,J_1))$. Every node in $f((i_1+1,J_2))$ must be equal or adjacent to v_2 and v_3, and v_2 is adjacent to v_3, so $f((i_1+1,J_2)) = v_1$ which contradicts uniformity. (We did suppose there was a i_2 with $f((i_1,1)) = f((i_2,1))$, so $f^{-1}(v_1) \supseteq \{(i_1,J_1)\} \cup \{(i_1+1,J_2)\} \cup \{(i_2,1)\}\})$. □

Now every node in $f((i_1+1,J_2))$ must be adjacent or equal to v_2 and to $v_3 = f((i_1+1,J_1))$. Note that v_3 is adjacent to v_1 and, because H does not contain 3-cycles or 4-cycles we have: $f((i_1+1,J_2)) = \{v_1\}$ and $f^{-1}(v_1) \supseteq \{(i_1+1,J_2),\ (i_1,J_1),\ (i_2,1)\}$, hence $|f^{-1}(v_1)| \geq c_2+1$, which contradicts uniformity. This completes the proof of the claim that there are no $i_1\neq i_2$ with $f((i_1,1)) = f((i_2,1))$. This shows that f_1 restricted to the set of nodes $\{(i,1)|1\leq i\leq n\}$ is a graph isomorphism of a cycle with n nodes to a subgraph of $H=H_0$, so H_0 has a Hamiltonian circuit.

Suppose H_0 has a Hamiltonian circuit. First note that $f\colon V_G \to \{0,\ldots,n-1\}$, given by $f((i,j)) = i$ is a uniform emulation of V_G on a cycle with n nodes. Because uniform emulation is transitive, it is sufficient to give a uniform emulation of a cycle with n nodes on H for showing that G can be uniformly emulated on H.

First we derive a cyclic path in H that visits each node at least once and at most c_2 times: add to the Hamiltonian circuit of H_0 extra path parts that visit the branches, added to the nodes of H. This path can be transformed to a uniform emulation of the cycle with n nodes to H, by mapping successive nodes on the cycle on the same node v in H, if the path visits v less than c_2 times. □

Corollary 7.2.1.3. For every $c \in N^+$, the following problem is NP-complete:

[c-UNIFORM EMULATION]

Instance: Connected graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, with $|V_G|=c|V_H|$.

Question: Is there a uniform emulation of G on H?

By further refining the technique of the proof of theorem 7.2.1.2. slightly better results can be obtained.

Theorem 7.2.1.4. For every c_1, $c_2 \in N^+$ with $c_2 \geq 2$ the following problem is NP-complete:

Instance: Connected, undirected, planar graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, with each node of V_G of degree at most 3, and each node of V_H of degree at most c_2+2, and $|V_G|=c_1 \cdot c_2 \cdot |V_H|$.

Question: Is there a uniform emulation of G on H?

Note: the theorem is stronger than theorem 7.2.1.2. in two ways: we can choose G and H planar, and the degree of the nodes in G is not dependent on c_1.

Proof. Clearly the problem is in NP. To prove NP-completeness we will again transform HAMILTONIAN CIRCUIT for undirected planar graphs with nodes of degree 3 to this problem [GJ79,GJT76]. As noted before, this version of HAMILTONIAN CIRCUIT is NP-complete.

Let $H_0=(V_0,E_0)$ be an arbitrary undirected planar graph with nodes of degree 3. We construct a graph $H=(V_H,E_H)$ by adding to each node of V_0 c_2-1 extra branches, each branch now consisting of $2c_1+1$ nodes. An example is given in fig. 7.2.1.3., with $c_1=3$ and $c_2=2$. Note that $|V_0|$ must be even (every node in H_0 has exactly 3 adjacent edges), hence $|V_H|$ is even. Let $n=|V_H|$.

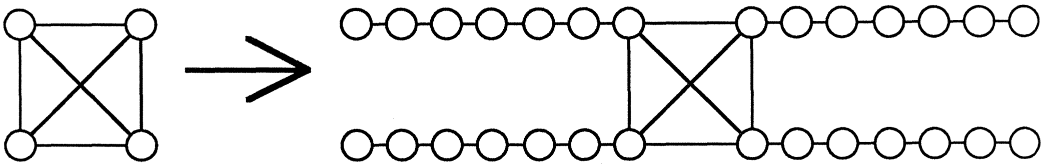


fig. 7.2.1.3. H_0 and H with $c_1=3$ and $c_2=2$.

Let $G=(V_G,E_G)$ be defined by $V_G = \{(i,j) \mid 0 \le i \le n-1 \wedge 1 \le j \le c_1\}$, and $E_G = \{((i_1,j_1), (i_2,j_2)) \mid (i_1,j_1),(i_2,j_2) \in V_G \wedge ((i_1=i_2 \wedge |j_1-j_2| =1 \wedge (\min(j_1,j_2) \text{ is odd} \Leftrightarrow i_1 \text{ is odd})) \vee (j_1=j_2 \wedge i_1=i_2 \pm 1 \pmod{n}))\}$. Thus the cliques of the graph G of the proof of theorem 7.2.1.2. are replaced by structures that have a much smaller number of edges. The example of fig. 7.2.1.4. illustrates the construction. In this example n=6 and $c_1=5$.

Again we claim that H_0 contains a Hamiltonian circuit if and only if there is a uniform emulation of G on H. The proof of claim 7.2.1.2.1. for the case $c_2 \ge 2$ can be followed to obtain a proof for this case,

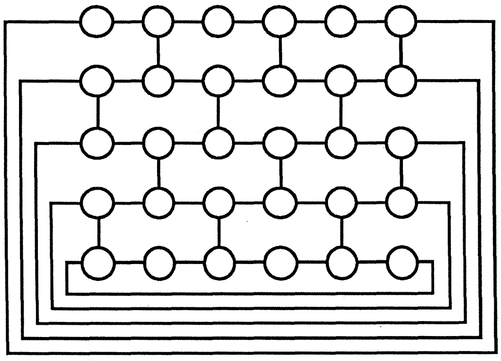


fig. 7.2.1.4. G with n=6 and $c_1=5$.

with the following observation: for every $v \in V_0$ and every branch of v there is at least one node that is mapped upon the last node of the branch (i.e. the node with degree 1). For every j, $1 \leq j \leq c_1$ there is a node $(i,j) \in V_G$ that has a distance $\leq 2c$ to w. This means that (i,j) must also be mapped upon a node of this branch, so the image of the cycle $A_j = \{(i,j) | 0 \leq i \leq n-1\}$ must visit this branch. Now the remaining part of this proof can be done in a similar way as in the proof of claim 7.2.1.2.1. Notice that for every n, c_1 G is planar. If H_0 has a Hamiltonian circuit, then one can construct a uniform emulation of G on H similar to the construction in claim 7.2.1.2.1. □

We mention the following corollaries of theorem 7.2.1.2 and 7.2.1.4:

Corollary 7.2.1.5. c-UNIFORM EMULATION FOR PLANAR GRAPHS is NP-complete.

Corollary 7.2.1.6. For every $c \in N^+$, c even, the following problem is NP-complete:

> Instance: Connected, undirected, planar graphs $G = (V_G,E_G)$ and $H = (V_H,E_H)$, with each node of G of degree at most 3, each node of H of degree at most 4, and $|V_G| = c \cdot |V_H|$.
>
> Question: Is there a uniform emulation of G on H?

Proof. This is a special case of theorem 7.2.1.4. with $c_1=c/2$ and $c_2=2$. □

Corollary 7.2.1.7. For every $c \in N^+$ the following problem is NP-complete:

> Instance: Connected, undirected, planar graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, with each node of G of degree at most 2 (i.e. G is a path of a cycle), each node of H of degree at most c+2, and $|V_G|=c \cdot |V_H|$.
>
> Question: Is there a uniform emulation of G on H?

Proof. This is a special case of theorem 7.2.1.2. with $c_1=1$ and $c_2=c$. Note that the graphs H, resulting from the construction in the proof of theorem 7.2.1.2. are planar. □

The result of corollary 7.2.1.7. should be contrasted with the following proposition:

Proposition 7.2.1.8. For every $c \in N^+$, and connected, undirected graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$ with G a cycle or a path (i.e., each node of V_G has degree at most 2), each node of H of degree at most c and $|V_G|=c \cdot |V_H|$, there exists a uniform emulation of G on H. The emulation function can be found in $O(|V_G|+|E_H|)$ time.

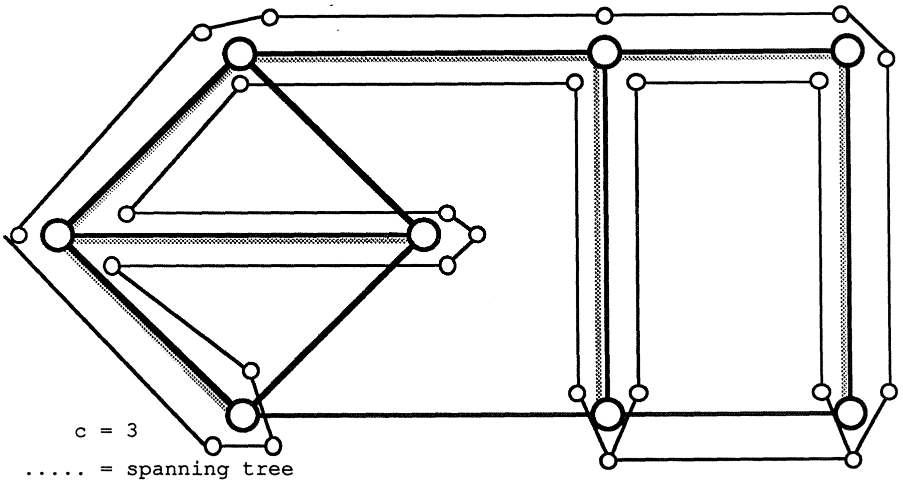
Proof. It is easy to construct a cyclic path that visits each node at least once and at most c times: construct a spanning tree T of H. Now visit the root of T; visit recursively the nodes in the subtree of the leftmost son of the root, starting and ending with the root of this subtree, then visit the root of T, then visit the nodes in the subtree of the one-but-leftmost son of the root, then again visit the root of T, etc. In this way a node v is visited at least once, and at most degree(v) $\le$ c times.

We can map the nodes of the path or cycle G on the successive nodes visited by this algorithm. If the algorithm visits a node v less than c times we have to map some successive nodes of G on v. In this way a uniform emulation can be obtained.

The time necessary to create the spanning tree is $O(|E_H|)$, the rest of the work can be done in time $O(|V_G|)$. □

Interesting open problems are: given a computation factor c, what is the complexity of the problem to determine whether a path or a cycle can be uniformly emulated on a graph with maximum node degree c+1; and is the c-UNIFORM EMULATION problem for G and H graphs with maximum node degree 3 NP-complete (for c>1)?

7.2.2. Directed graphs of bounded degree. For directed graphs the c-UNIFORM EMULATION problem remains NP-complete if we restrict G to be a cycle and H a graph with each node involved in at most 3 edges:



c = 3

..... = spanning tree

----- = uniform emulation of a cycle on H

fig. 7.2.1.5

Theorem 7.2.2.1. For every c ∈ N^+ the following problem is NP-complete:

Instance: Directed, strongly connected, planar graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, such that every node in G is involved in at most 2 edges (i.e., G is a cycle), every node in H is involved in at most 3 edges, and $|V_G|=c\cdot|V_H|$.

Question: Is there a uniform emulation of G on H?

Proof. Clearly the problem is in NP. To prove NP-completeness we transform DIRECTED HAMILTONIAN CIRCUIT for directed, strongly connected, planar graphs with each node involved in exactly 3 edges to this problem. This version of DIRECTED HAMILTONIAN CIRCUIT is NP-complete [GJ79].

Let a directed, strongly connected, planar graph $H_0=(V_0,E_0)$ be given, with each node $v \in V_0$ involved in exactly 3 edges. H_0 has two types of nodes. (See fig. 7.2.2.1.)

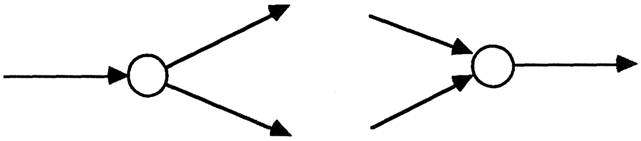


fig. 7.2.2.1.

We first replace H_0 by H_1 by replacing nodes as in fig. 7.2.2.2. (The digits 0, 1, 2 indicate the "type" of a new node for later reference.)

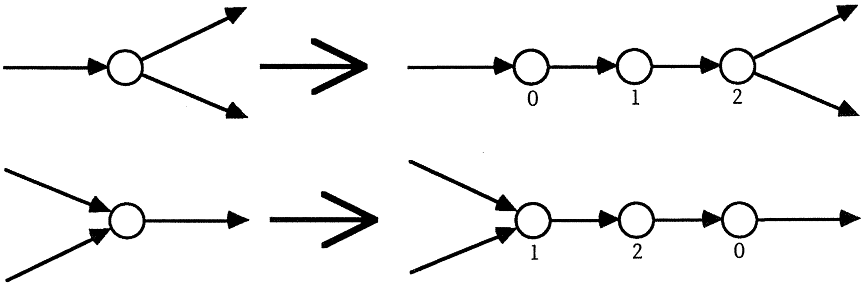


fig. 7.2.2.2.

To each node of type 2 in fig. 7.2.2.2. we add a binary tree with c-1 leaves, with the edges in the tree directed towards the leaves. Another tree with edges directed towards the root is placed on these leaves. The root of this tree is connected with an edge to the corresponding node of type 1. An example is given in fig. 7.2.2.3., with c=7. An example of the whole transformation is given in fig. 7.2.2.4. (with c=3).

Let $H=(V_H,E_H)$ be the graph that is obtained from H_1 in this way. If c=1 then one can take $H=H_0$. We let $n=|V_H|$, and G a directed cycle of c·n

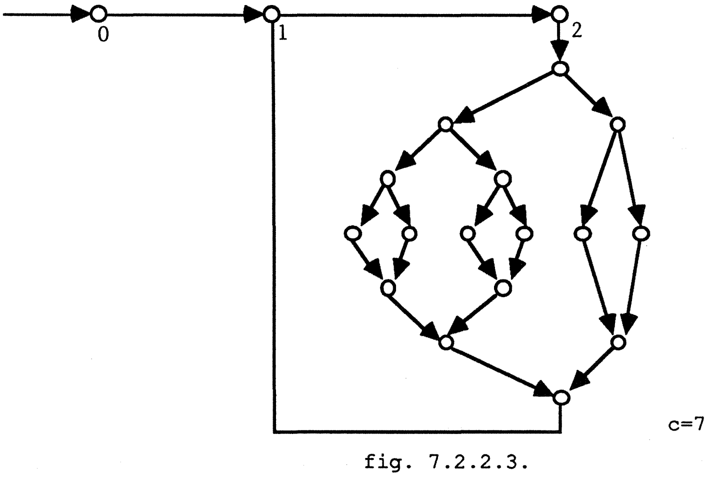


fig. 7.2.2.3.

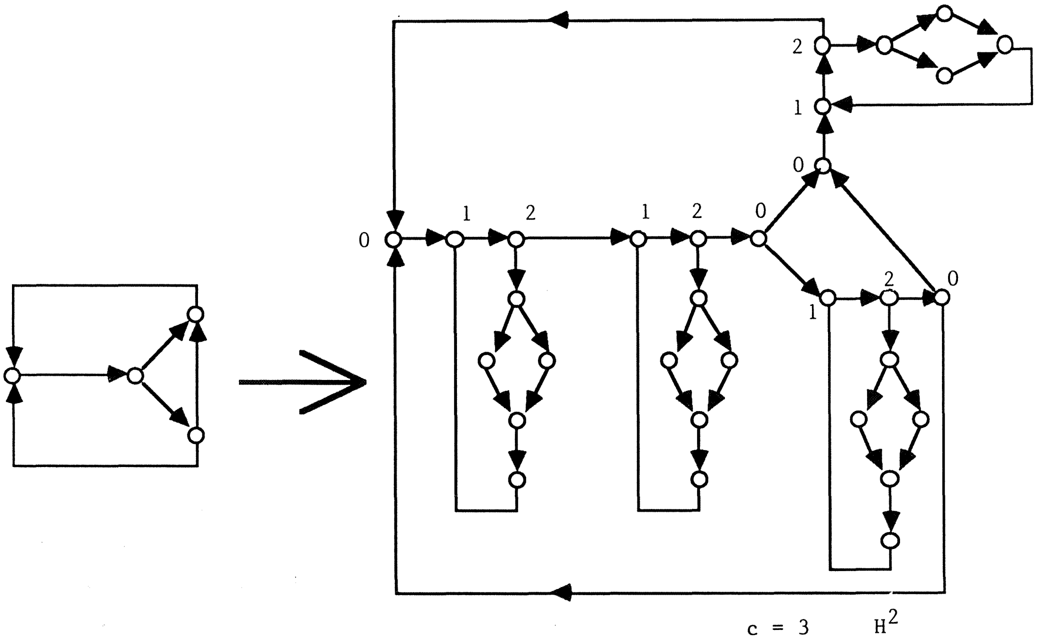


fig. 7.2.2.4.

nodes.

Claim 7.2.2.1.1. H_0 contains a Hamiltonian circuit if and only if there is a uniform emulation of G on H.

Proof. If H_0 contains a Hamiltonian circuit then there exists a cyclic path in H that visits each node at least once and at most c times. We visit nodes of type 0 in the order of the Hamiltonian circuit and, when we arrive at a pair of nodes of type 1 and 2, we go c-1 times over the added structure of the two trees, each time visiting another leaf. This path can be transformed to a uniform emulation of G on H by mapping successive nodes of G on the same node v in H, if the path visits v less than c times.

Now suppose we have a uniform emulation f of G on H. For every node v* of type 1, look at the c predecessors of the c nodes that are mapped upon v* by f. Because every leaf in the two-tree structure added to v*, (and the neighbor of v* of type 2) must be visited by f(G), and there are c-1 such leaves, c-1 of these predecessors must be mapped upon the root of the tree with edges towards v* and exactly one is mapped upon a node of type 0 or 2. This means that every node cluster, consisting of a node of type 0, a node of type 1, a node of type 2 and the added two-tree structure can be visited at most once from another node cluster. It has also to be visited at least once from another node cluster. Look at the successive node clusters in H that are visited. The corresponding nodes in H_0 form a Hamiltonian circuit. □

The graphs G and H fulfill the conditions and can be obtained from H_0 in time polynomial in $|V_0|$ and c. This completes the proof of theorem 7.2.2.1. □

7.3. The complexity of finding uniform emulation on networks of a certain type. In this section we consider the c-UNIFORM EMULATION problem for the case that the host graph H is required to be a network of some given type, for the following types of networks: the two-dimensional grid, the cube network, the shuffle-exchange network and the 4-pin shuffle. In section 7.3.1. we prove that c-UNIFORM EMULATION is NP-complete if the host graph is restricted to be a grid, for every $c\geq2$. In section 7.3.2. we prove that c-UNIFORM EMULATION is NP-complete if the host graph is restricted to be a cube, for every $c\geq1$. In section 7.3.3. we prove that c-UNIFORM EMULATION is NP-complete if the host graph is restricted to be a 4-pin shuffle or a shuffle-exchange graph, for every $c\geq7$ and $c\geq15$.

7.3.1. Uniform emulation on the two-dimensional grid network. Recall the definitions of the two-dimensional grid network with and without wrap-around connections from section 1.6.2.

Theorem 7.3.1.1. For every $c \in N^+$, $c\geq2$, the following problem is NP-complete:

[c-UNIFORM EMULATION ON A GRID]

Instance: A connected, undirected graph $G=(V_G,E_G)$, such that there is an $n \in N^+$ with $|V_G|=c\cdot n^2$.

Question: Is there a uniform emulation of G on GR_n?

Proof. Clearly the problem is in NP. To prove NP-completeness we first consider the following problem.

Given a set of $V \subseteq V_n$ in a two-dimensional grid $GR_n=(V_n,E_n)$, the subgraph of GR_n induced by V is the graph $G_V=(V,E_V)$, with $E_V=\{(v_1,v_2)|v_1,v_2 \in V \text{ and } (v_1,v_2) \in E_n\}$, i.e. every two nodes in V that are adjacent in the grid are adjacent in the subgraph induced by V. The following problem is known to be NP-complete [IPS84]:

[HAMILTONIAN CIRCUIT IN A GRID GRAPH]

Instance: $n \in N$ and a set of nodes $V \subseteq V_n$.

Question: Does the subgraph of GR_n, induced by V contain a Hamiltonian circuit?

We show that HAMILTONIAN CIRCUIT IN A GRID GRAPH can be polynomially transformed to c-UNIFORM EMULATION ON A GRID (for every $c\geq2$). Let $n \in N^+$ and a set of nodes $V \subseteq V_n$ be given and let $c\geq2$, $c \in N^+$. We will construct a connected undirected graph $G=(V_G,E_G)$, with $|V_G|=c\cdot(2n)^2$, such that there is a uniform emulation of G on GR_{2n}, if and only if the subgraph of GR_n induced by V contains a Hamiltonian circuit.

We let $G=(V_G,E_G)$ consist of the following parts:

a) c-1 grid layers of 2nx2n nodes:

$A = \{v^i_{j,k}|1\leq i\leq c-1,\ 0\leq j\leq 2n-1,\ 0\leq k\leq 2n-1\}$.

b) one grid layer of 2nx2n nodes with the nodes of V omitted:

$B = \{v^c_{j,k}|0\leq j\leq 2n-1,\ 0\leq k\leq 2n-1,\ (j,k) \notin V\}$.

c) a cycle of |V| points:

$C = \{w_i|0\leq i\leq |V|-1\}$.

We connect points whose (j,k)-coordinates are adjacent in GR_{2n}, without regard to the layer, and one node of the cycle to some point $v^i_{j,k}$ with $(j,k) \in V$. Choose an arbitrary $(j^*,k^*) \in V$.

Now $V_G = A \cup B \cup C$;

$$E_G = \{(\ v^{i_1}_{j_1,k_1},\ v^{i_2}_{j_2,k_2}\ )|\ v^{i_1}_{j_1,k_1},\ v^{i_2}_{j_2,k_2} \in A \cup B \wedge ((j_1,k_1), (j_2,k_2)) \in E_{2n}\} \cup \{(w_i,\ w_j)|w_i,w_j \in V \wedge i=(j\pm1) \bmod |V|\} \cup \{(w_1,v^1_{j^*,k^*})\}.$$ (E_{2n} is the set of edges of GR_{2n}.)

Claim 7.3.1.1.1. The subgraph of GR_n induced by V contains a Hamiltonian circuit if and only if there is a uniform emulation of $G=(V_G,E_G)$ on GR_{2n}.

Proof. First suppose the subgraph of GR_n induced by V contains a Hamiltonian circuit. Then the subgraph of GR_{2n} induced by V contains a Hamiltonian circuit. Now let $f(v^i_{j,k}) = (j,k)$, for all $v^i_{j,k} \in A \cup B$. Then every node in V_{2n}-V has c nodes of $A \cup B$ mapped upon it, every node in V has c-1 such nodes mapped upon it. Now we can map w_1 on (j^*,k^*), w_2 on the node that is visited by the Hamiltonian circuit after (j^*,k^*), etc., i.e., we let f map w_i on the i'th node on the Hamiltonian circuit, where (j^*,k^*) is considered to be the first node. In this way a correct uniform emulation of G on GR_{2n} is obtained.

Now suppose f is a uniform emulation of $G=(V_G,E_G)$ on GR_{2n}. We claim that for all $(j_1,k_1) \in \{0,\dots,2n-2\} \times \{0,\dots,2n-2\} - \{0,\dots,n\} \times \{0,\dots,n\}$, $(j_2,k_2) \in \{0,\dots,2n-1\} \times \{0,\dots,2n-1\}$, $1 \le i_1,i_2 \le c$, with $(j_1,k_1) \ne (j_2,k_2)$: $f(v^{i_1}_{j_1,k_1}) \ne f(v^{i_2}_{j_2,k_2})$. Suppose there do exist (j_1,k_1), (j_2,k_2), i_1,i_2 fulfilling the conditions, with $f(v^{i_1}_{j_1,k_1}) = f(v^{i_2}_{j_2,k_2})$. Then we can reach a contradiction as follows: $V \subseteq \{0,\dots,n-1\} \times \{0,\dots,n-1\}$, so $(j_1,k_1) \notin V$ and (j_1,k_1) is not a neighbor of a node in V. So $v^{i_1}_{j_1,k_1}$ has 5c-1 neighbors in G*, and $v^{i_2}_{j_2,k_2}$ has at least one neighbor in G* that is not $v^{i_1}_{j_1,k_1}$ or a neighbor of $v^{i_1}_{j_1,k_1}$, so at least 5c+1 nodes must be mapped upon $f(v^{i_1}_{j_1,k_1})$ and its 4 neighbors in GR_{2n}. This contradicts uniformity. In the same way one can prove that for (j_1,k_1), i_1 as before, and $0 \le i \le |V|-1$, $f(w_i) \ne f(v^{i_1}_{j_1,k_1})$. So the only nodes that can be mapped to $f(v^{i_1}_{j_1,k_1})$ ($(j_1,k_1) \in \{0\dots 2n-2\} \times \{0\dots 2n-2\} - \{0\dots n\} \times \{0\dots n\}$), are the nodes $v^{\alpha}_{j_1,k_1}$, $1 \le \alpha \le c$. There are exactly c such nodes, so for all $(j_1,k_1) \in \{0\dots 2n-2\} \times \{0\dots 2n-2\} - \{0\dots n\} \times \{0\dots n\}$, i_1, $i_2 \in \{1,\dots,c\}$: $f(v^{i_1}_{j_1,k_1}) = f(v^{i_2}_{j_2,k_2})$. Now we claim that the set $\{v^1_{i,2n-2} | i=0...2n-1\}$ must be mapped upon a cycle in GR_{2n} (that is a set of the form $\{(j,k) | o\leq k\leq 2n-1\}$ or $\{(k,j) | 0\leq k\leq 2n-1\}$, with j fixed). Suppose this is not the case. Then the successive nodes $f(v^1_{i,2n-2})$ (i=0...2n-1) form a path that must make a bend on the grid. Suppose there is a bend in the form as shown in fig. 7.3.1.1. (The other cases are similar.) Then the nodes $\{f(v^1_{i*,2n-3}) | i* \in\{i,i+1,i+2\}\}$ cannot be mapped such that adjacencies are preserved. Contradiction.

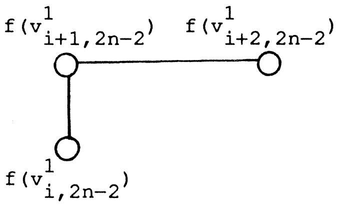


fig. 7.3.1.1.

In the same way one can prove that the set $\{v^1_{2n-2,i} | i=0...2n-1\}$ must be mapped upon a cycle in GR_{2n}. Without loss of generality one may suppose $f(v^1_{i,2n-2}) = (i,2n-2)$ and $f(v^1_{2n-2,i}) = (2n-2,i)$. With induction one can prove that for all (i,j) $f(v_{i,j}) = (i,j)$. Now $f(v^{\alpha}_{i,j}) = (i,j)$ for all $v^{\alpha}_{i,j} \in A \cup B$. This means that $f(w_i) \in V$ for all $w_i \in C$ (use uniformity of f). So $f(w_1)$, $f(w_2),...,f(w_{|V|})$ form a Hamiltonian circuit in G_V. □

Note that $|V_G| = c\cdot(2n)^2$, and the construction of G can be carried out in time polynomial in |V| and n. Hence c-UNIFORM EMULATION ON A GRID is NP-complete, for c≥2. □

Theorem 7.3.1.2. For every $c \in N^+$, c≥2 the following problem is NP-complete:

Instance: A connected, undirected graph $G=(V_G,E_G)$, such that there is an $n\in N^+$ with $|V_G|=c\cdot n^2$.

Question: Is there a uniform emulation of G on GR'_n?

Proof. Similar to that of theorem 7.3.1.1. □

7.3.2. Uniform emulation on the cube network. In this section we use the definition of the cube network of section 1.6.4. Recall from theorem 6.2.2. that GR_{2^n} can be uniformly emulated on C_{2n}, i.e., GR_{2^n} is isomorphic to a spanning subgraph of C_{2n}. The main result in this section is:

Theorem 7.3.2.1. For every $c \in N^+$, the following problem is NP-complete:

[c-UNIFORM EMULATION ON A CUBE]

Instance: A connected, undirected graph $G=(V_G,E_G)$, such that there is a $k \in N^+$ with $|V_G|=c\cdot 2^k$.

Question: Is there a uniform emulation of G on C_k?

Proof. Clearly the problem is in NP. To prove NP-completeness we will transform the HAMILTONIAN CIRCUIT IN A GRID GRAPH problem to this problem. (For details on this version of HAMILTONIAN CIRCUIT see the proof of theorem 7.3.1.1.)

Let $n \in N^+$ and a set of nodes V in GR_n be given. We may suppose that $n=2^k$ for some $k \in N^+$. (If not we can reduce the problem to this case in polynomial time.)

Let f be a uniform emulation of GR_n on C_{2k} (as implied from theorem 6.2.2.) (Note that f is a bijection, i.e. a subgraph isomorphism.) Let g be the mapping $C_{2k} \rightarrow C_{6k+2}$, defined by $g(x_1\ldots x_{2k}) = x_1x_1x_1x_2x_2x_2\ldots x_{2k}x_{2k}x_{2k}oo$.

Let $V_1 = g\circ f(V)$, and

let $V_2 = \{x_1x_1x_1x_2x_2x_2\ldots x_{i-1}x_{i-1}x_{i-1}o01x_{i+1}x_{i+1}x_{i+1}\ldots x_{2k}x_{2k}x_{2k}oo \mid$
$x_1\ldots x_{i-1}ox_{i+1}\ldots x_{2k},\ x_1\ldots x_{i-1}1x_{i+1}\ldots x_{2k} \in f(V)\}$,

$V_3 = \{x_1x_1x_1x_2x_2x_2\ldots x_{i-1}x_{i-1}x_{i-1}o11x_{i+1}x_{i+1}x_{i+1}\ldots x_{2k}x_{2k}x_{2k}oo \mid$
$x_1\ldots x_{i-1}ox_{i+1}\ldots x_{2k},\ x_1\ldots x_{i-1}1x_{i+1}\ldots x_{2k} \in f(V)\}$,

$V_4 = \{x_1x_1x_1x_2x_2x_2\ldots x_{2k}x_{2k}x_{2k}o1 \mid f^{-1}(x_1\ldots x_{2k}) \in V$ and has at most 3 neighbors in V}

$V_5 = \{x_1x_1x_1x_2x_2x_2\ldots x_{2k}x_{2k}x_{2k}1o \mid f^{-1}(x_1\ldots x_{2k}) \in V$ and has at most 2 neighbors in V}

We let $W = V_1 \cup V_2 \cup V_3 \cup V_4 \cup V_5 \subseteq V_{6k+2}$ and G_W be the subgraph of C_{6k+2} induced by W: $G_W=(W,E_W)$ and $E_W=\{(v,w) \mid v,w \in W \wedge (v,w) \in E_{6k+2}\}$ (where E_{6k+2} is the set of edges of C_{6k+2}). G_W resembles the subgraph of GR_n induced by V, $G_V=(V,E_V)$, with $E_V = \{(v,w) \mid v,w \in V$ and v,w adjacent in $GR_n\}$. One can obtain a graph isomorphic to G_W from G_V by adding on each edge 2 extra nodes (these correspond to the nodes of V_2 and V_3), by adding to each node in V with 3 neighbors one extra neighbor (these correspond to nodes of V_4) and by adding to each node in V with 2 neighbors 2 extra neighbors (nodes of V_4 and V_5). In fig. 7.3.2.1. we show an example of this transformation. Although G_W is a subgraph of a (6k+2)-cube and not of a grid, we draw it as a subgraph of a grid, for convenience.

Let $G=(V_G,E_G)$, with $V_G=\{v_{i,j} \mid 0 \leq i \leq |V|-1$ and $1 \leq j \leq 5\}$, and $E_G=\{(v_{i,1},v_{i,j}) \mid v_{i,1},v_{i,j} \in V_G$ and $2 \leq j \leq 5\} \vee \{(v_{i,2},v_{i+1,5}) \mid v_{i,2}, v_{i+1,5} \in V_G\}$, where + is the addition modulo |V|. For example, if |V|=6, then G is the graph of fig. 7.3.2.2. Notice $|V_G|=5 \cdot |V|=|W|$.

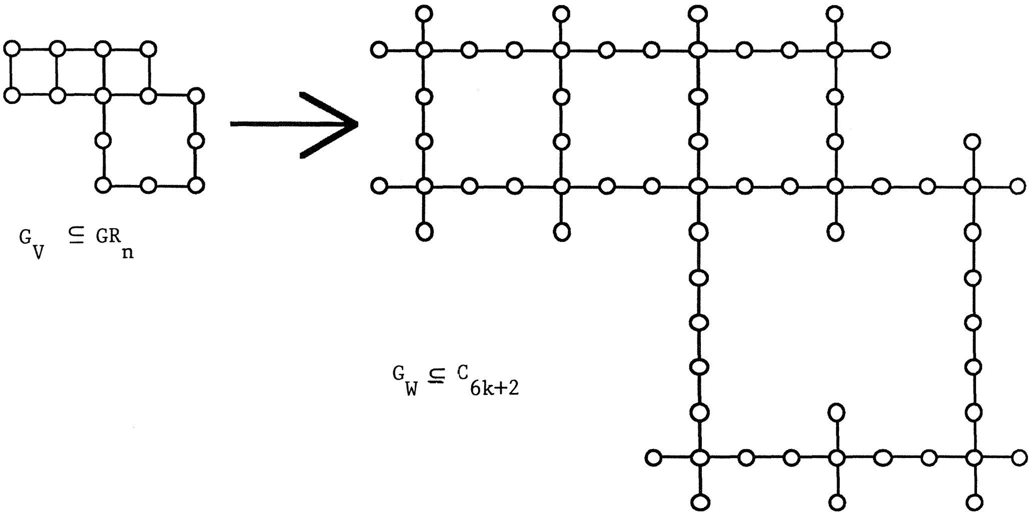
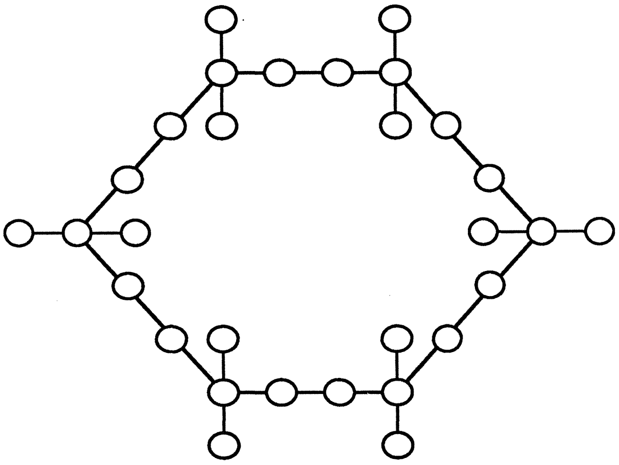


fig. 7.3.2.1.



G, with |V|=6.

fig. 7.3.2.2.

Claim 7.3.2.1.1. G is isomorphic to a (spanning) subgraph of G_W if and only if G_V has a Hamiltonian circuit.

Proof. Let G_V have a Hamiltonian circuit. Number the successive nodes visited by this circuit $v_0, v_1, \ldots, v_{|V|-1}$. So $v_i \in V$ and v_i is adjacent to $v_{(i+1)\mathrm{mod}|V|}$. Now we can give a subgraph isomorphism $\tilde{f}$ of G into G_W. Let $\tilde{f}(v_{i,1}) = g \circ f(v_i)$, for all i, $0 \leq i \leq |V|-1$. We now map the nodes $v_{i,2}$ and $v_{i+1,5}$ on the two nodes between $g \circ f(v_i)$ and $g \circ f(v_{i+1})$. (This is a node in V_2 and a node in V_3.) There are two other nodes adjacent to $g \circ f(v_i)$ and we map $v_{i,3}$ and $v_{i,4}$ on these nodes. In this way an isomorphism of G to a (spanning) subgraph of G_W is obtained.

If $\tilde{f}$ is an isomorphism of G to a (spanning) subgraph of G_W then consider the row $\tilde{f}(v_{0,1}), \tilde{f}(v_{1,1}), \ldots, \tilde{f}(v_{|V|-1,1})$. Each of these nodes $\tilde{f}(v_{i,1})$ must have at least 4 neighbors in G_W, so is of the form $x_1x_1x_1x_2x_2x_2 \ldots x_{2k}x_{2k}x_{2k}oo$ and there is a node w of GR_{2k} with $\tilde{f}(v_{i,1}) = g \circ f(w)$. Let $w_0, \ldots, w_{|V|-1}$ be the nodes in GR_{2k} such that $g \circ f(w_i) = v_{i,1}$ for all i, $0 \leq i \leq |V|-1$. w_i is adjacent to $w_{(i+1)\mathrm{mod}|V|}$ (this is because $\tilde{f}(v_{i,2})$ and $\tilde{f}(v_{(i+1)\mathrm{mod}|V|,5})$ must be adjacent), so $w_0, w_1, \ldots, w_{|V|-1}$ form a Hamiltonian circuit in $V \subseteq GR_n$. □

Let $m=6k+2$. To complete the proof we will use basically the same technique as in section 4: we take a graph G' with the property that, for every emulation f of G' on C_{m-1}, if f maps at most c nodes of G' upon each node of C_{m-1}, f maps $c-1$ nodes on the nodes of W and c nodes on the nodes of $C_m \backslash W$. To this graph we add G, which will have to be mapped upon the nodes of W.

Choose a $x' \in (\frac{o}{1})^{m+1}$, such that $x'_1 \ldots x'_m \in H$ and $x'_{m+1}=1$, and $x'_1 \ldots x'_m$ has degree 4 in G_H. Now let $G_\star=(V_\star,E_\star)$ with $V_\star=V_G \cup \{v^i_{x_1 \ldots x_{m+1}} \mid x_1 \ldots x_{m+1} \in (\frac{o}{1})^{m+1}$ and $1 \leq i \leq c$ and ($i \neq c$ or $x_1 \ldots x_m \notin W$ or $x_{m+1} = 1)\}$ and $E_\star = E_G \cup \{(v^i_x, v^j_y) \mid v^i_x, v^j_y \in V_\star - V_G$ and x is adjacent to y in $C_{m+1}\} \cup \{(v_{0,1}, v^c_{x'})\}$.

Notice that $G_\star$ is connected and $|V_\star|=c \cdot 2^{m+1}$.

Claim 7.3.2.1.2. $G_\star$ can be uniformly emulated on C_{m+1} if and only if G is isomorphic to a (spanning) subgraph of G_W.

Proof. If h is an isomorphism of $G_\star$ to a (spanning) subgraph of G_W, then we can suppose without loss of generality that $h(v_{0,1})=x'$. (Notice that $h^{-1}(x')$ must be a node $v_{i,1}$ for some i, $0 \leq i \leq |V|-1$, and use the symmetry of G.) Now let $f(v_{i,j}) = h(v_{i,j}) \cdot o$ for all $v_{i,j} \in V_G$ and $f(v^i_x) = x$ for all $v^i_x \in V_\star - V_G$. One can easily check that f is a uniform emulation of $G_\star$ on C_{m+1}.

Now suppose f is a uniform emulation of G^* on C_{m+1}. We will show that G is isomorphic to a subgraph of G_W. We first need:

Claim 7.3.2.1.2.1. $\forall i,j$, $1 \leq i,j \leq c$ $\forall x_1 \ldots x_m \in (\frac{o}{1})^m$ $f(v^i_{x_1 \ldots x_m 1}) = f(v^j_{x_1 \ldots x_m 1})$.

Proof. $f(v^i_{x_1 \ldots x_m 1})$ and $f(v^j_{x_1 \ldots x_m 1})$ are equal or adjacent. If they are adjacent then every node $f(v^\alpha_{x_1 \ldots \bar{x}_{i_0} \ldots x_m 1})$ ($1 \leq \alpha \leq c$, $1 \leq i_0 \leq m$) must be adjacent or equal to $f(v^i_{x_1 \ldots x_m 1})$ and to $f(v^j_{x_1 \ldots x_m 1})$. Because C_{m+1}

does not contain triangles (cycles with length 3), each of these nodes must be equal to $f(v^i_{x_1\ldots x_m 1})$ or to $f(v^j_{x_1\ldots x_m 1})$. So at least mc+2 nodes are mapped upon the 2 nodes $f(v^i_{x_1\ldots x_m 1})$, $f(v^j_{x_1\ldots x_m 1})$. This contradicts uniformity. Hence $f(v^i_{x_1\ldots x_m 1}) = f(v^j_{x_1\ldots x_m 1})$. □

<u>Definition</u>. For $0 \le p \le n$, a p-face of C_n is any subgraph of 2^p nodes of C_n that have identical bits in n-p corresponding positions (i.e.: a p-face of C_n is a subgraph of C_n, isomorphic to C_p).

Consider $A = \{v^1_{x_1\ldots x_m 1} \mid x_1\ldots x_m \in (\frac{o}{1})^m\}$. There do not exist $v,w \in A$, $v \ne w$ and $f(v)=f(w)$, else at least 2c nodes are mapped upon $f(v)=f(w)$ (use claim 7.3.2.1.2.1.) This means that f(A) is isomorphic to A, so is an m-face of C_{m+1}. Without loss of generality we may suppose $f(v^1_{x_1\ldots x_m 1}) = x_1\ldots x_m 1$ for all $x_1\ldots x_m \in (\frac{o}{1})^m$. Now for all i, $1 \le i \le c$, $x_1\ldots x_m \in (\frac{o}{1})^m$ $f(v^i_{x_1\ldots x_m 1}) = x_1\ldots x_m 1$. For all i, $1 \le i \le c$, $x_1\ldots x_m \in (\frac{o}{1})^m$ with $v^i_{x_1\ldots x_m o} \in V_*$ we have that $f(v^i_{x_1\ldots x_m o})$ must be adjacent or equal to $f(v^i_{x_1\ldots x_m 1}) = x_1\ldots x_m 1$. Due to the uniformity of f, $f(v^i_{x_1\ldots x_m o})$ cannot be mapped upon a node in $\{y_1\ldots y_m 1 \mid y_1\ldots y_m \in (\frac{o}{1})^m\}$, so $f(v^i_{x_1\ldots x_m o}) = x_1\ldots x_m o$.

Let $B = \{x_1\ldots x_m o \mid x_1\ldots x_m \in W\}$. We now have that each node in $V_{m+1}\backslash B$ has c nodes of V_*-V_G mapped upon it by f and each node in B has c-1 nodes of V_*-V_G mapped upon it by f. So the nodes of V_G must be mapped upon nodes of B; f, restricted to V_G is a bijection of V_G to B. Because f must preserve adjacencies and G_B (the subgraph of C_{m+1} induced by B) is graph isomorphic to G_W, G is isomorphic to a (spanning) subgraph of G_W. The function ψ: $x \rightarrow f(x)|_m$ is a graph isomorphism $V_G \rightarrow W$. □

Combining claim 7.3.2.1.1. and 7.3.2.1.2. and noting that every construction can be done in time polynomial in |V|, we have a polynomial time transformation from the HAMILTONIAN CIRCUIT IN A GRID GRAPH problem to the c-UNIFORM EMULATION ON A CUBE-problem. Hence the latter is NP-complete. □

Corollary 7.3.2.2. The following problem is NP-complete:

[SUBGRAPH ISOMORPHISM IN A CUBE GRAPH]

Instance: A connected, undirected graph $G=(V_G,E_G)$ and a set of nodes $W \subseteq C_n$

Question: Is G isomorphic to a subgraph of G_W, the subgraph of C_n induced by W?

7.3.3. Uniform emulation on the 4-pin shuffle and the shuffle-exchange network. In this section we use the definitions of the shuffle-exchange network and the 4-pin shuffle, given in section 1.6.3. and the notations introduced in section 1.5.

Theorem 7.3.3.1. For every $c \in N^+$, with $c \geq 7$ the following problem is NP-complete:

[c-UNIFORM EMULATION ON A 4-PIN SHUFFLE]

Instance: A directed strongly connected graph $G=(V_G,E_G)$, such that there is an $n \in N^+$ with $|V_G|=c \cdot 2^n$.

Question: Is there a uniform emulation of G on S_n?

Proof. Clearly the problem is in NP. To prove NP-completeness we will transform HAMILTONIAN CIRCUIT for directed graphs with each node involved in exactly 3 edges to this problem. As noted in section 7.2.2., this version of HAMILTONIAN CIRCUIT is NP-complete [GJ79]. Let $c \geq 7$ be given. Note that c is the computation factor of the uniform emulation.

Let a directed graph $G=(V,E)$ be given with each node involved in exactly 3 edges. We will construct a strongly connected graph $G^*=(V^*,E^*)$ such that G^* can be uniformly emulated on S^n (where $c \cdot 2^n=|V^*|$), if and only if G contains a Hamiltonian circuit. Without loss of generality we may suppose that G is strongly connected with no self-loops or parallel edges.

Let α be the smallest integer such that $2^\alpha-1 \geq |V|$. Let f_1 be an injection of V to the set $\{x_1 \ldots x_\alpha | \forall i \; x_i=o \vee x_i=1 \wedge \exists i \; x_i=o\} = (\frac{o}{1})^\alpha \backslash \{1^\alpha\}$. Such an injection can easily be found in time polynomial in

$|V|$. Let $n=5\alpha+7$. Let f_2 be the mapping $(\frac{o}{1})^{\alpha} \to (\frac{o}{1})^{n}$, given by $f_2(x) = o1^{\alpha}oxo1^{\alpha+1}oxo1^{\alpha}o$.

Let $W \subseteq (\frac{o}{1})^{n}$ be given by $W = \{y|$ y is a substring of length n of a string $o1^{\alpha}oxo1^{\alpha+1}oxo1^{\alpha}oyo1^{\alpha+1}oyo1^{\alpha}o$, where there are $v, w \in V$, with $(v,w) \in E$ and $x=f_1(v)$ and $y \in f_1(w)\}$. With the subgraph of S_n induced by W we denote the graph $G_W=(W,E_w)$, with $E_W = \{(v,w)|\ (v,w) \in E_n,\ v,w \in W\}$.

Now we claim:

<u>Claim 7.3.3.1.1</u>. G has a Hamiltonian circuit, if and only if there is a cycle of $|V|\cdot(4\alpha+5)$ nodes in G_W, that visits each node of W at most once.

<u>Proof</u>. First suppose V has a Hamiltonian circuit. Let $v_1,\ldots,v_{|V|}$ be the successive nodes on this circuit.

For every pair of nodes v_i, v_{i+1} (+ is addition modulo $|V|$) notice that every substring of length n of the string $o1^{\alpha}of_1(v_i)o1^{\alpha+1}of_1(v_i)o1^{\alpha}of_1(v_{i+1})o1^{\alpha+1}of_1(v_{i+1})o1^{\alpha}o$ is an element of W. So there is a path of length $4\alpha+5$ of $f_2 \circ f_1(v_i)$ to $f_2 \circ f_1(v_{i+1})$. By adding these paths together one obtains a cycle in G_W of length $|V|\cdot(4\alpha+5)$. It is not difficult to check from the construction of f_1, f_2 and W that no node of W can be visited more than once, in this cycle.

Now suppose there is a cycle of $|V|.(4\alpha+5)$ nodes in G_W that visits each node of W at most once.

Note that every address of a node of W has exactly one substring $1^{\alpha+1}$. When we follow the cycle this substring moves in the addresses of the nodes we visit one place to the left, i.e. we visit successively nodes of the form $(\frac{o}{1})^{n-(\alpha+1)-m}1^{\alpha+1}(\frac{o}{1})^{m}$, $(\frac{o}{1})^{n-(\alpha+1)-m-1}1^{\alpha+1}(\frac{o}{1})^{m+1}$, etc. After a node of the form $1^{\alpha+1}(\frac{o}{1})^{n-(\alpha+1)}$ a node of the form $(\frac{o}{1})^{n-(\alpha+1)}1^{\alpha+1}$ is visited. This means that every $(4\alpha+5)$ steps on the cycle a node of the form $(\frac{o}{1})^{2\alpha+3}1^{\alpha+1}(\frac{o}{1})^{2\alpha+3}$ is visited. Necessarily this node is of the form $o1^{\alpha}oxo1^{\alpha+1}oxo1^{\alpha}o \in f_2 \circ f_1(V)$.

Now let $v_1,\ldots,v_{|V|}$ be the nodes of V, such that the cycle successively visits $f_2 \circ f_1(v_1), \ldots, f_2 \circ f_1(v_{|V|})$ after every $4\alpha+5$ steps. There is a path of length $4\alpha+5$ from $f_2 \circ f_1(v_i)$ to $f_2 \circ f_1(v_{i+1})$; due to the construction of f_1, f_2, and W there now must be an edge $(v_i, v_{i+1}) \in E$. (+ is the addition modulo $|V|$.) So $v_1, \ldots, v_{|V|}$ form a Hamiltonian circuit. □

Now we will give the definition of $G^*=(V^*,E^*)$. G^* consists of the following parts: we take c-1 layers, each consisting of a copy of S_n. Between the layers there are connections between copies of the same nodes and copies of adjacent nodes. From the upper two layers we leave out the nodes that are a member of W. To this we add $R_{|V|(4\alpha+5)}[K_2]$: that is two cycles with length $|V| \cdot (4\alpha+5)$, with connections between copies of the same and adjacent nodes; and one cycle with just the right number of nodes to fill up all the remaining free places.

Definitions.

$V_1 = \{v_{x,i} \mid x \in (\frac{o}{1})^n,\ 1 \le i \le c-3\}$

$\quad \subseteq \{v_{x,i} \mid x \in (\frac{o}{1})^n,\ x \notin W,\ i=c-2 \vee i=c-1\}.$

$V_2 = \{w_{i,j} \mid o \le i \le (4\alpha+5).|V|-1,\ j=1 \vee j=2\}.$

$r = c.2^n - |V_1| - |V_2|.$

$V_3 = \{z_i \mid o \le i \le r-1\}.$

$V^* = V_1 \cup V_2 \cup V_3$. Let an arbitrary $v^* \in V$ be given.

$$\begin{aligned}
E^* = {} & \{(v_{x,i}, v_{y,j}) \mid v_{x,i}, v_{y,j} \in V_1 \wedge ((x,y) \in E_n \vee (x=y \wedge i \ne j))\} \\
& \cup \{(w_{i_1,j_1}, w_{i_2,j_2}) \mid (i_1=i_2 \wedge j_1 \ne j_2) \vee (i_2=i_1+1 \bmod (4\alpha+5)|V|) \\
& \qquad \text{and } w_{i_1,j_1}, w_{i_2,j_2} \in V_2\} \\
& \cup \{(z_i, z_{(i+1) \bmod r}) \mid z_i, z_{(i+1) \bmod r} \in V_3\} \\
& \cup \{(v_{f_2 \circ f_1(v^*),1}, w_{0,1}), (w_{0,1}, v_{f_2 \circ f_1(v^*),1}), (v_{0^n,1}, z_0), (z_0, v_{0^n,1})\}.
\end{aligned}$$

One may notice that $G^*=(V^*,E^*)$ is strongly connected, and $|V^*|=c \cdot 2^n$.

Claim 7.3.3.1.2. There exists a uniform emulation of G^* on S_n, if and only if there is a cycle of length $|V| \cdot (4\alpha+5)$ in G_W, that visits each node of W at most once.

Proof. First suppose there is a cycle of length $|V| \cdot (4\alpha+5)$ in G_W, that visits each node of W at most once. We will now construct a uniform emulation f of G* on S_n. For $v_{x,i} \in V_1$ we let $f(v_{x,i})=x$. In this way we have mapped c−1 nodes of V_1 on every node in $V_1 \backslash W$ and c−3 nodes on every node in W. From our previous observations it is clear that the cycle must use every node of the form $f_2 \circ f_1(v)$, for some $v \in V$, so it must use also $f_2 \circ f_1(v^*)$. We can now map the nodes of V_2 on the nodes of W, in the following manner: $f(w_{0,1}) = f(w_{0,2}) = f_2 \circ f_1(v^*)$. Let w^i be the i'th node after $f_2 \circ f_1(v^*)$ on the cycle. Then $f(w_{i,1}) = f(w_{i,2}) = w^i$. In this way adjacencies are preserved and every node in $(\frac{o}{1})^n$ has c−1 or c−3 nodes of $V_1 \cup V_2$ mapped upon it. We now use that S_n has a Hamiltonian circuit (this follows from the existence of binary the Bruyn sequences [dB46]). The successive nodes of V_3 are mapped on the successive nodes of this Hamiltonian circuit, starting with $f(z_0)= o^n$. However when we arrive in a node that has c−3 nodes of $V_1 \cup V_2$ mapped upon it, we map 3 successive nodes of V_3 on this node. In this way a uniform emulation of G* on S_n is obtained. □

Now suppose there exists a uniform emulation f of G* on S_n.

Claim 7.3.3.1.2.1. For all $v_{x,i}, v_{x,j} \in V_1$ $f(v_{x,i}) = f(v_{x,j})$.

Proof. Suppose there are $v_{x,i}, v_{x,j} \in V_1$ with $f(v_{x,i}) \neq f(v_{x,j})$. Note that $(v_{x,i}, v_{x,j}) \in E^*$ and $(v_{x,j}, v_{x,i}) \in E^*$. So $(f(v_{x,i}), f(v_{x,j})) \in E_n$ and $(f(v_{x,j}), f(v_{x,i})) \in E_n$, hence $\{f(v_{x,i}), f(v_{x,j})\} = \{(o1)^*[o], (1o)^*[1]\}$.

Every node $v_{\frac{o}{1}x_1 \ldots x_{n-1},k} \in V_1$ must be mapped adjacent or equal to $f(v_{x,i})$ and to $f(v_{x,j})$, so $f(v_{\frac{o}{1}x_1 \ldots x_{n-1},k}) \in \{(o1)^*[o], (1o)^*[1]\}$. In the same way one shows that for all $v_{x_2 \ldots x_n \frac{o}{1},k} \in V_1$ $f(v_{x_2 \ldots x_n \frac{o}{1},k}) \in \{(o1)^*[o], (1o)^*[1]\}$. If $x \in \{o^n, 1^n, (o1)^*[o], (1o)^*[1]\}$, then notice that neither these nodes and neither their neighbors are members of W, so we have at least 3(c−1)>2c nodes mapped upon $\{(o1)^*[o], (1o)^*[1]\}$, which contradicts uniformity. If $x \notin \{o^n, 1^n, (o1)^*(o), (1o)^*[1]\}$, then all 5 nodes $\{\frac{o}{1}x_1 \ldots x_{n-1}, x_1 \ldots x_n, x_2 \ldots x_{n-1}\frac{o}{1}\}$ are different, so there are at least 5(c−3) nodes mapped upon the 2 nodes {(o1)*[o], (1o)*[1]}, again contradicting uniformity. □

Claim 7.3.3.1.2.2. For all $v_{x,i} \in V_1$ $f(v_{x,i}) = x$
or for all $v_{x,i} \in V_1$ $f(v_{x,i}) = \bar{x}$.

Proof. If $x \neq y$ then $f(v_{x,1}) \neq f(v_{y,1})$. (Suppose $f(v_{x,1}) = f(v_{y,1})$. Then at least $2(c-3) > c$ nodes are mapped upon one node, contradicting uniformity. Here we use $c \geq 7$, and claim 7.3.3.1.2.1.). So the mapping $x \rightarrow f(v_{x,1})$ is an isomorphism of S_n onto itself. Using lemma 5.3.2.4. we now have that either for all $x \in ({o \atop 1})^n$ $f(v_{x,1}) = x$ or for all $x \in ({o \atop 1})^n$ $f(v_{x,1}) = \bar{x}$. With claim 7.3.3.1.2.1. the result follows. □

Without loss of generality we may suppose $f(v_{x,i}) = x$ for all $v_{x,i} \in V_1$. Now we have shown that on nodes of $({o \atop 1})^n \backslash W$ there are c−1 nodes of V_1 that are mapped upon that node, and on nodes of W there are c−3 nodes that are mapped upon that node by f.

Claim 7.3.3.1.2.3. For all i, $0 \leq i \leq (4\alpha+5) \cdot n-1$ $f(w_{i,1}) = f(w_{i,2})$.

Proof. $f(w_{i,1})$ and $f(w_{i,2})$ must be adjacent to each other. It is easy to check that $f(w_{i,1})$, $f(w_{i,2}) \notin$ {(o1)*[o], (1o)*[1]} ((o1)*[o], (1o)*[1] are not elements of W and neither are their neighbors; now use uniformity). Hence $f(w_{i,1}) = f(w_{i,2})$. □

Now we have that uniformity forces that every node of V_2 is mapped upon a node of W. Furthermore, if $0 \leq i,j \leq (4\alpha+5)|V|-1$, $i \neq j$ and $f(w_{i,1}) = f(w_{j,1})$ then a contradiction with uniformity arises. This shows that the successive nodes $f(w_{i,1})$, $i=0\ldots(4\alpha+5)\cdot|V|-1$ are mapped upon the successive nodes of a cycle of length $(4\alpha+5)\cdot|V|$ in G_W, that visits each node of W at most once. □

Corollary 7.3.3.1.3. G* can be uniformly emulated on S_n if and only if G has a Hamiltonian circuit.

The construction of G* can be done in polynomial time in |V|, hence the problem stated in theorem 7.3.3.1. is NP-complete. □

By using lemma 5.2.8. one easily obtains the following corollary of theorem 7.3.3.1.

Corollary 7.3.3.2. For every $c \in N^{+}$, $c \geq 7$ the following problem is NP-complete:

[c-UNIFORM EMULATION ON AN INVERSE 4-PIN SHUFFLE]

Instance: A directed, strongly connected graph $G=(V_G,E_G)$, such that there is an $n \in N^{+}$ with $|V_G|=c\cdot 2^{n}$.

Question: Is there a uniform emulation of G on IS_n?

Theorem 7.3.3.3. For every $c \in N$, $c \geq 15$ the following problem is NP-complete:

[c-UNIFORM EMULATION ON A SHUFFLE-EXCHANGE GRAPH]

Instance: A directed, strongly connected graph $G=(V_G,E_G)$, such that there is an $n \in N^{+}$ with $|V_G|=c\cdot 2^{n}$.

Question: Is there a uniform emulation of G on SE_n?

Proof. The proof is more or less similar to that of theorem 7.3.3.1. Clearly the problem is in NP. To prove NP-completeness we transform HAMILTONIAN CIRCUIT for directed graphs with each node involved in exactly 3 edges to this problem.

Let $c \geq 15$ be given and let a directed graph $G=(V,E)$ with each node involved in exactly 3 edges be given. We may suppose that G is strongly connected. Again let α be the smallest integer such that $2^{\alpha}-1 \geq |V|$ and let f_1 be an injection of V to the set $(\frac{o}{1})^{\alpha}\setminus\{1^{\alpha}\}$. Let $n=12\alpha+6$. Let f_2 be the mapping $(\frac{o}{1})^{\alpha} \rightarrow (\frac{o}{1})^{n}$ given by $f_2(x) = o1^{2\alpha}ox\bar{x}o1^{4\alpha+1}ox\bar{x}o1^{2\alpha}$. Let $W_1 = \{y|$ y is a substring of length n of a string $o1^{2\alpha}ox\bar{x}o1^{4\alpha+1}ox\bar{x}o1^{2\alpha}oy\bar{y}o1^{4\alpha+1}oy\bar{y}o1^{2\alpha}$, where there are $v,w \in V$ with $(v,w) \in E$ and $x=f_1(v)$ and $y \in f_1(w)\}$. Let $W = W_1 \cup \{y|\ y_1 \ldots y_{n-1}\bar{y}_n \in W_1\}$. Let the subgraph of SE_n, induced by W be the graph $G_W=(W,E_W)$, with $E_w = \{(v,w)|\ (v,w) \in \tilde{E}_n, v,w \in W\}$. G_W has the property that between every pair of nodes $f_2 \circ f_1(w_1)$ and $f_2 \circ f_1(w_2)$, with $(w_1,w_2) \in E$ there is a path that uses exactly $10\alpha+5$ shuffle-edges and $4\alpha+2$ exchange edges of SE_n. One can prove in a way similar to claim 7.3.3.1.1.:

Claim 7.3.3.3.1. G contains a Hamiltonian circuit if and only if there is a cycle of $|V| \cdot (14\alpha+7)$ nodes in G_W, that visits each node of W at most once.

We will now define $G^*=(V^*,E^*)$, such that G^* can be emulated on SE_n, if and only if G has a Hamiltonian circuit. G^* consists of the following parts: we take c-2 layers, each consisting of a copy of SE_n, with connections between the layers between copies of the same node and copies of adjacent nodes, but from the upper five layers we leave out the nodes that are a member of W. To this we add $R_{|V| \cdot (14\alpha+7)}[K_5]$, (that is: five cycles with connections between the layers between copies of the same node and between copies of adjacent nodes), and -again- one cycle with just the right number of nodes to fill up all the remaining free places.

Definitions.

$V_1 = \{v_{x,i} \mid x \in (\frac{o}{1})^n,\ 1 \le i \le c-7\}$

$\quad \cup \{v_{x,i} \mid x \in (\frac{o}{1})^n,\ x \notin W,\ c-6 \le i \le c-2\}.$

$V_2 = \{w_{i,j} \mid 0 \le i \le (14\alpha+7) \cdot |V|-1,\ 1 \le j \le 5\}.$

$r = c \cdot 2^n - |V_1| - |V_2|.$

$V_3 = \{z_i \mid 0 \le i \le r-1\}.$

$V^* = V_1 \cup V_2 \cup V_3$. Let an arbitrary $v^* \in V$ be given.

$E^* = \{(v_{x,i}, v_{y,j}) \mid v_{x,i}, v_{y,j} \in V_1 \wedge (x,y \in \tilde{E}_n \vee (x=y \wedge i \ne j))\}$

$\quad \cup \{(w_{i_1,j_1}, w_{i_2,j_2} \mid (i_1=i_2 \wedge j_1 \ne j_2) \vee (i_2=i_1+1 \bmod (14\alpha+7) \cdot |V|)$

$\qquad \wedge (w_{i_1,j_1}, w_{i_2,j_2} \in V_2)\}$

$\quad \cup \{(z_i, z_{(i+1) \bmod r}) \mid z_i, z_{(i+1) \bmod r} \in V_3\}$

$\quad \cup \{(v_{f_2 \circ f_1(v^*),1}, w_{0,1}), (w_{0,1}, v_{f_2 \circ f_1(v^*),1}), (v_{o^n,1}, z_0), (z_0, v_{o^n,1})\}.$

Again $G^*=(V^*,E^*)$ is strongly connected and $|V^*|=c \cdot 2^n$.

Claim 7.3.3.3.2. There exists a cyclic path in SE_n that visits each node of SE_n at least once and at most twice.

Proof. We use the fact that there exists a Hamiltonian circuit in S_{n-1}. Let $b^0, b^1, b^2, \ldots, b^{2^{n-1}-1}$ be the successive nodes on this circuit. The following algorithm generates the desired path: (let + and − be addition and subtraction modulo 2^{n-1}).

```
For i := 0 to $2^{n-1}$
do begin visit $b^i\, b_1^{i-1}$
         if $b_1^{i-1} = b_{n-1}^{i+1}$ then visit $b^i\, \overline{b_1^{i-1}} = b^i \overline{b_{n-1}^{i+1}}$
         visit $b^i\, b_{n-1}^{i+1}$
   end
```

It is clear that this algorithm visits each node at least once and at most twice. Furthermore note that $b^{i+1}b_1^{i-1}$ can be obtained from $b^i b_{n-1}^{i+1}$ from one cyclic shift, so the successive nodes indeed form a path in SE_n. □

Claim 7.3.3.3.3. There exists a uniform emulation of G* on SE_n if and only if there is a cycle of length $|V|\cdot(14\alpha+7)$ in G_W that visits each node of W at most once.

Proof. If there is a cycle of length $|V|\cdot(14\alpha+7)$ in G_W that visits each node of W at most once, then we can map the nodes of V_1 and V_2 on V_n in the same way as in the proof of claim 7.3.3.1.3. Notice that in this way each node has at most c−2 nodes of $V_1 \cup V_2$ mapped upon it. We can use the nodes of V_3 to fill up the remaining free places, using the circular path of claim 7.3.3.3.2.

Now suppose there exist a uniform emulation f of G* on S_n. We first need:

Claim 7.3.3.3.3.1. For all $v_{x,i}$, $v_{x,j} \in V_1$ $f(v_{x,i}) = f(v_{x,j})$.

Proof. Suppose there are $v_{x,i}$, $v_{x,j} \in V_1$ with $f(v_{x,i}) \neq f(v_{x,j})$. Then $f(v_{x,i})$ and $f(v_{x,j})$ must be mapped upon mutually adjacent nodes. By observing adjacencies one shows that every node of the form $v_{x_1\ldots x_n,k}$, $v_{x_2\ldots x_n x_1,k}$, $v_{x_n x_1\ldots x_{n-1},k}$, or of the form $v_{x_1\ldots x_{n-1}\bar{x}_n,k}$ in V_1 must be mapped upon $f(v_{x,i})$ or $f(v_{x,j})$. So at least $4(c-7) > 2c$ nodes are mapped upon 2 nodes, contradicting uniformity. □

Claim 7.3.3.3.3.2. Either for all $v_{x,i} \in V_1$ $f(v,i)=x$ or for all $v_{x,i} \in V_1$ $f(v_{x,i})=\bar{x}$.

Proof. Similar to 7.3.3.1.2.2. □

Claim 7.3.3.3.3.3. For all i, $0 \le i \le (14\alpha+7) \cdot |V|-1$, k_1,k_2, $1 \le k_1,k_2 \le 5$ $f(w_{i,k_1}) = f(w_{i,k_2})$.

Proof. Suppose there exists an i, $0 \le i \le (14\alpha+7) \cdot |V|-1$, such that $f(w_{i,k_1}) \ne f(w_{i,k_2})$ (for some k_1,k_2, $1 \le k_1,k_2 \le 5$). Then $f(w_{i,k_1})$ and $f(w_{i,k_2})$ are mutually adjacent nodes. Since there do not exist pairs of mutually adjacent nodes b,c in SE_m, such that there is a node d in SE_m with $(b,d) \in \tilde{E}_m$ and $(c,d) \in \tilde{E}_m$ or $(d,b) \in \tilde{E}_m$ and $(d,c) \in \tilde{E}_m$ we have that $f(w_{i-1,j}) \in \{f(w_{i,k_1}), f(w_{i,k_2})\}$ $(1 \le j \le 5)$, $f(w_{i+1,j}) \in \{f(w_{i,k_1}), f(w_{i,k_2})\}$ $(1 \le j \le 5)$, and $f(w_{i,j}) \in \{f(w_{i,k_1}), f(w_{i,k_2})\}$ $(1 \le j \le 5)$. (+ and − are addition and subtraction modulo $(14\alpha+7) \cdot |V|$.) There are also at least c−7 nodes of V_1 mapped upon each node of SE_m (see claim 7.3.3.3.3.2.), so $|f^{-1}(\{w_{i,k_1}, w_{i,k_2}\})| \ge 2(c-7)+15 > 2c$. This contradicts uniformity. □

In the same way as in the proof of claim 7.3.3.1.2. we can show that the successive nodes $f(w_{i,1})$ $(0 \le i \le (14\alpha+7) \cdot |V|-1)$ form a cycle in G_W that visits each node of W at most once, of length $(14\alpha+7) \cdot |V|$. □

So G* can be uniformly emulated on SE_n if and only of G contains a Hamiltonian circuit. The construction of G* can be done in time, polynomial in |V|, hence c-UNIFORM EMULATION ON A SHUFFLE EXCHANGE GRAPH is NP-complete, for c≥15. □

Corollary 7.3.3.4. For every $c \in N$, c≥15 the following problem is NP-complete:

[c-UNIFORM EMULATION ON AN INVERSE SHUFFLE-EXCHANGE GRAPH]

Instance: A directed, strongly connected graph $G=(V_G,E_G)$, such that there is an $n \in N^+$ with $|V_G|=c \cdot 2^n$.

Question: Is there a uniform emulation of G on ISE_n?

We conjecture that also for smaller computation factors c the problems stay NP-complete.

7.4. The complexity of finding uniform emulations on paths and ring networks.

7.4.1. Introduction. In this section we consider UNIFORM EMULATION for the case that H is a path or a ring (cycle). We will call these problems UNIFORM EMULATION ON A PATH and UNIFORM EMULATION ON A RING, respectively. We use the definitions of the path graph, and the bandwidth and cyclic bandwidth of a graph from section 1.5. and the definition of the ring from section 1.6.1.

The UNIFORM EMULATION ON A PATH-problem is strongly related with the problem to determine whether a graph G has a bandwidth of at most K, for some $K \in N^+$. To be precise, the BANDWIDTH problem is the following:

[BANDWIDTH]

Instance: A graph $G=(V_G,E_G)$ and an integer $K \leq |V_G|$

Question: Is there a bijection $f\colon V_G \rightarrow \{0,\ldots,|V_G|-1\}$ such that for all $(v,w)\in E_G$ $|f(v)-f(w)|\leq K$, i.e. is bandwidth(G) $\leq$ K?

Leung, Vornberger and Witthoff [LVW84] introduced a variant of BANDWIDTH, called CYCLE-BANDWIDTH. This is the problem to decide whether a graph G has cyclic bandwidth K or less. The CYCLE-BANDWIDTH problem is strongly related with the UNIFORM EMULATION ON A RING-problem. The relation of the UNIFORM EMULATION ON A PATH (RING) with the various types of bandwidth problems was already suggested by lemma 5.2.4. - 5.2.7.

This section is organized as follows. In section 7.4.3. we prove UNIFORM EMULATION ON A PATH and ON A RING NP-complete. In section 7.4.4. we will show that this result even holds, when the guest graph G is chosen to be a tree, with each node degree at most 3. In section 7.4.5. NP-completeness results are given for the directed variants of the problems. In section 7.4.6. we will give polynomial time algorithms for the UNIFORM EMULATION ON A PATH (RING)-problems, for the case the computation factor (that is, $|V_G|$ divided by the length of the path (ring)) is some fixed constant. In section 7.4.7. we will show that the problem becomes again NP-complete, if we keep the requirement that the computation factor is fixed, but drop the requirement that G is connected, for each computation factor $c \geq 4$ (UNIFORM EMULATION ON A PATH) or each computation factor $c \geq 2$ (UNIFORM EMULATION ON A RING).

7.4.2. NP-completeness for arbitrary undirected graphs. The problem to determine whether a given , connected graph G=(V,E) has bandwidth K or less is NP-complete, even when we restrict G to trees with no node degree exceeding 3 [GJ79]. On the other hand, when we fix the bandwidth factor K to a constant there exists an algorithm that is polynomial in $|V|$, (but exponential in K) to determine whether a given graph G has bandwidth at most K [Sa80]. CYCLIC BANDWIDTH however is NP-complete even for fixed $K \geq 2$, if we allow the graphs to be not connected. If G is required to be connected then CYCLIC BANDWIDTH can be solved in polynomial time for any fixed K [LVW84]. (It is not difficult to provide a proof that the problem whether a given connected graph has cyclic bandwidth K or less is NP-complete (for variable K), by transformation from BANDWIDTH.)

The resemblance of these results to the results for UNIFORM EMULATION ON A PATH and UNIFORM EMULATION ON A RING, respectively, is not very surprising, given lemmas 5.2.4. and 5.2.5., but the details are tedious.

Theorem 7.4.2.1. The following problem is NP-complete:

[UNIFORM EMULATION ON A PATH]

Instance: A connected graph G = (V,E) and an $n \in N^+$

Question: Is there a uniform emulation of G on P_n?

Proof. Clearly the problem is in NP. To prove NP-completeness we will transform BANDWIDTH to this problem. Let a graph G= (V_G,E_G) and a $K \in N^+$ be given. Let n= $|V_G|$. We will construct a graph G′ = (V_1,E_1) such that G′ can be uniformly emulated on P_{2n+5} if and only if G has bandwidth $\leq$ K. One may suppose that G is connected.

Let c = 4 K^2+4K+4. We will now define G′. (We will only give a more or less informal description of G'). We let G' consist of the following parts:

a) 2 "blocks": subgraphs with nodes $B^{\alpha}=\{v_{i\ j}^{\alpha} | 1\leq i\leq 3, \ 1\leq j\leq c\}$ $(\alpha=1,2)$ and edges $\{(v_{i_1,j_1}^{\alpha}\ ,\ v_{i_2,j_2}^{\alpha}) | v_{i_1,j_1}^{\alpha},\ v_{i_2,j_2}^{\alpha} \in B^{\alpha},\ v_{i_1,j_1}^{\alpha} \neq v_{i_2,j_2}^{\alpha}$ and $|i_1-i_2| \leq 1\}$. (The blocks will have to be mapped on the 'ends' of P_{2n+5}.)

b) a path with length 2n-1 between the two blocks: nodes $w_o,w_1\ldots,w_{2n-2}$; with connections between w_i and w_{i+1} $(o\leq i\leq 2n-3)$; w_o is connected to $v_{3,j}^{1}$ for all j, $1\leq j\leq c$ and w_{2n-1} is connected to $v_{1,j}^{2}$ for all j, $i\leq j\leq c$. (This path must be mapped on a one-to-one basis to the nodes 3,..., 2n+1.)

c) we add to each node w_i with i odd $\frac{3}{4}c$ $(=3K^2 +3K)$ nodes which will have to be mapped upon the same node as w_i: i.e. nodes $\{w_{i,j} | 1\leq i\leq 2n-3$, i odd and $1\leq j\leq\frac{3}{4}c\}$ and $w_{i,j}$ is connected to w_{i-1} and w_{i+1}.

d) for every node $x\in G$ we let G' have a clique of $\frac{3}{4}c$ nodes, that 'represents x': for every $x\in V_G$ we have nodes $\{(x)_i | i=1\ldots\frac{3}{4}c\}$, and edges (x_i,x_j) for $i \neq j$.

e) for every pair of nodes $\{x,y\}$ with $(x,y)\in E_G$ we connect the clique representing x and the clique representing y by a path of length 2K-1. We have nodes $\{x,y\}_i$ $o\leq i\leq 2K-2$ with $\{x,y\}_1$ connected to every node x_j, $1\leq j\leq\frac{3}{4}c$ and $\{x,y\}_{2K-1}$ connected to every node y_j, $1\leq j\leq\frac{3}{4}c$ and $\{x,y\}_i$ connected to $\{x,y\}_{i+1}$. (The notation is somewhat imprecise. Of course $(x,y)\in E_G \Leftrightarrow (y,x)\in E_G$. We will however have one, and not two paths between the clique representing x and the clique, representing y.)

f) We make the graph connected by choosing an arbitrary $x\in V_G$, and connecting $(x)_1$ to w_{2n-2} by a path with length 2n-1, (so we use nodes $\tilde{v}_i$, $o\leq i\leq 2n-2$, $\tilde{v}_i$ connected to $\tilde{v}_{i+1}$, $\tilde{v}_o$ connected to w_{2n-2} and $\tilde{v}_{2n-2}$ connected to $(x)_1$).

g) Let the total number of nodes we used in parts a) - f) be m. It is easy to verify that $m<(2n+5)c$. Add a path with $(2n+5)c-m$ nodes to v_o, i.e. nodes $\tilde{w}_i$ $(o\leq i\leq (2n+5)c -m-1)$, $\tilde{w}_i$ connected to $\tilde{w}_{i+1}$ and $\tilde{w}_o$ connected to v_o. Now G' has $(2n+5)c$ nodes.

Let G' be the graph in this way obtained.

Claim 7.4.2.1.1. G' can be uniformly emulated on P_{2n+5}, if and only if G has bandwidth $\leq K$.

Proof. First let a linear ordering $f\colon V_G \rightarrow V_n$ with bandwidth $\leq K$ be given. We now construct a uniform emulation g of G' on P_{2n+5}. We will successively designate the nodes in V_{2n+5} on which the nodes of G' are mapped upon. When we have - on a certain moment in our construction - already mapped $c-\alpha$ nodes on a node $p \in V_{2n+5}$, then we say that p has α free places left.

a) $g(v^1_{i,j}) = i-1$ $(1 \leq i \leq 3,\ 1 \leq j \leq c)$
   $g(v^2_{i,j}) = 2n+1+i$ $(1 \leq i \leq 3,\ 1 \leq j \leq c)$

b) $g(w_i) = i+3$ $(o \leq i \leq 2n-2)$

c) $g(w_{i,j}) = i+3$ $(1 \leq i \leq 2n-3, i \text{ odd},\ 1 \leq j \leq \frac{3}{4}c)$

d) $g(\ (x)_i) = 2 \cdot f(x)+3$ $(x \in V_G,\ 1 \leq i \leq \frac{3}{4}c)$

Now we have mapped c nodes on the nodes $\{0, 1, 2, 2n+2, 2n+3, 2n+4\}$ and $\frac{3}{4}c+1$ nodes on every other node in V_{2n+5}.

e) If $(x,y) \in E_G$ and $|f(x)-f(y)| = K$ then the distance between the images of the node cliques representing x and y $(g(\{(x)_i | 1 \leq i \leq \frac{3}{4}c\})$ and $g(\{(y)_i | 1 \leq i \leq \frac{3}{4}c\}))$ is $2K$: we can map the nodes in the path $\{\{x,y\}_i | o \leq i \leq 2K-2\}$ on the nodes between $g((x)_1)$ and $g((y)_1)$. If $|f(x)-f(y)| < K$ then we map a part of the path on $2 \cdot \min(f(x),f(y))+3 = \min(g((x)_1),g((y)_1))$, and the rest of the path, in a one-to-one basis on the nodes between $g((x)_1)$ and $g((y)_1)$. In this way there are mapped at most $2+4+\ldots+2K-2 = K^2+K$ nodes from the paths of the form $\{x,y\}_i$, on a node in P_{2n+5}. So each node in $\{3,4,\ldots,2n+1\}$ has at least $\frac{1}{4}c-1-(K^2+K) = 3$ free places left.

f) By folding it once we can lay the path consisting of the v_i 's such that

- $g(\tilde{v}_o)$ is adjacent or equal to $g(w_{2n-2}) = 2n+1$.
- $g(\tilde{v}_{2n-1})$ is adjacent or equal to $g(x_1)$ for the chosen $x \in V_G$
- every node in H has at most 2 nodes of the path $\{v_i | i=1\ldots 2n-1\}$ mapped upon it
- $g(\{v_i |\ i=1\ldots 2n-1\}) \subseteq \{3,\ldots,2n+1\}$

At this moment each node in $\{3,\ldots, 2n+1\}$ has at least one free place.

g) With the path $\{\tilde{w}_i | o \leq i \leq c(2n+s)-m-1\}$ we can fill now every remaining free place and thus make the emulation uniform: let there be so far $c-r_1$ nodes mapped upon 3, then map the nodes $\tilde{w}_o, \ldots, \tilde{w}_{r_1-1}$ upon 3, let the number of nodes we have mapped so far upon 4 $c-r_2$, then map the nodes $\tilde{w}_{r_1} \ldots \tilde{w}_{r_1+r_2-1}$ upon 4, etc. In this way the path $\{\tilde{w}_i | i=o \ldots c(2n+5)-m-1\}$ can fill up every free place.

The mapping so obtained is a uniform emulation.

Now let g be a uniform emulation of G′ on P_{2n+5}. We will construct a linear ordering of G with bandwidth at most K.

First we claim that either $g(v^1_{i,j}) = i-1$ and $g(v^2_{i,j}) = 2n+1+i$ for all i,j, $1 \le i \le 3$, $1 \le j \le c$; or $g(v^2_{i,j}) = 3-i$ and $g(v^1_{i,j}) = 2n+5-i$ for all i,j, $1 \le i \le 3$, $1 \le j \le c$. To prove this, first note that nodes v^α_{2,j_1} and v^α_{2,j_2} have the same 3c−1 neighbors (with exception of the nodes self), and therefore must be mapped upon the same node. Now $g(v^\alpha_{i,j})$ and $g(v^\alpha_{i,j_2})$ for i=1 or i=3, $j_1 \neq j_2$ must be adjacent to $g(v^\alpha_{2,j_1})$ (due to uniformly they cannot be equal) and adjacent or equal to each other. So $g(v^\alpha_{i,j_1}) = g(v^\alpha_{i,j_2})$, for all $1 \le i \le 3$, $1 \le j_1, j_2 \le c$, $\alpha \in \{1,2\}$.

Each of the nodes in $g(\{v^\alpha_{i,j} | \alpha \in \{1,2\}, 1 \le i \le 3, 1 \le j \le c\})$ has c nodes of a block mapped upon it, so there are no other nodes (from b) -f)) mapped upon these nodes. Note that every node not in the blocks has a path of non-block nodes to the path $\{w_i | o \le i \le 2n-2\}$ (b)). This means that every node not in the blocks must be mapped between the images of the two blocks. Uniformity gives that the images of the blocks must lay upon the two ends of H. Because there is a path with 2n−1 nodes between $\{v^1_{3,j} | 1 \le j \le 3\}$ and $\{v^2_{1,j} | 1 \le j \le c\}$ we have $d_{P_{2n+5}}(g\{v^1_{3,j} | 1 \le j \le 3\}), g\{v^2_{1,j} | 1 \le j \le 3\}) \le 2n$, which shows our claim holds.

Without loss of generality we may suppose $g(v^1_{i,j}) = i-1$ and $g(v^2_{i,j}) = 2n+1+i$, for all i,j, $1 \le i \le 3$, $1 \le j \le c$. This forces $w_i = i+3$ for all i, $o \le i \le 2n-2$ and $w_{i,j} = i+3$ for all odd $i \in \{1,3,5,\ldots, 2n-3\}$ and j, $1 \le j \le \frac{3}{4}c$. So there are on each node in $\{o,1,2,2n+2,2n+3,2n+4\}$ c nodes not of the form $(x)_i$ for $x \in V_G$ mapped, on each node in $\{3,\ldots,2n+1\}$ that is odd numbered one node, and on each node in $\{3,\ldots,2n+1\}$ that is even numbered $\frac{3}{4}c+1$ nodes not of the form $(x)_i$ for $x \in V_G$ mapped. For every $x \in V_G$ $|g(\{(x)_i | 1 \le i \le \frac{3}{4}c\})| \le 2$ (all nodes in the set must be mutually adjacent). Because g is uniform, and our previous observation there is exactly <u>one</u> odd numbered node in $g(\{(x)_i | 1 \le i \le \frac{3}{4}c\})$, for all $x \in V_G$. This node we call $\tilde{g}(x)$. $\tilde{g}$ has the following properties:

1. $\tilde{g}(x)$ is odd, for all $x \in V_G$
2. $3 \le \tilde{g}(x) \le 2n-3$, for all $x \in V_G$
3. If $x \ne y$ then $\tilde{g}(x) \ne \tilde{g}(y)$, for all $x,y \in V_G$ (use uniformity).
4. If $(x,y) \in E_G$ then there is a path between $\{(x)_i | 1 \le i \le \frac{3}{4}c\}$ and $\{(y)_i | 1 \le i \le \frac{3}{4}c\}$ with length 2K-1: because g is an emulation we have that $|\tilde{g}(x)-\tilde{g}(y)| \le 2K$.

So the function $f: V_G \to V_n$, defined by $f(x) = (\tilde{g}(x)-3)/2$ is a bijection and $(x,y) \in E_G \Rightarrow |f(x)-f(y)| \le K$. So f is a linear ordering with bandwidth at most K.

Finally notice that the construction of G′ can be done in time polynomial in n. This completes the proof of theorem 7.4.2.1. □

<u>Theorem 7.4.2.2</u>. The following problem is NP-complete:

[UNIFORM EMULATION ON A RING]

<u>Instance</u>: An undirected, connected graph G and an integer $n \in N^+$.

<u>Question</u>: Is there a uniform emulation of G on R_n?

<u>Proof</u>. One can use the same construction as in 7.4.2.1. Rather than that the blocks must be mapped on the ends of the path, they must be mapped now adjacent to each other on the ring; the proof of claim 7.4.2.1.1. is easily translated to this case. □

The technique, used in the proof of theorem 7.4.2.1. will be used again in the next section. In section 7.5. we prove with a different technique a stronger result, namely that UNIFORM EMULATION is NP-complete, even if H is fixed to any connected graph, that is not complete. For instance, H can be fixed to a path or a ring with a fixed length.

7.4.3. NP-completeness for trees. We will now show that the NP-completeness results of section 7.4.2. also hold when G is restricted to trees with each node of degree at most 3.

Theorem 7.4.3.1. The following problem is NP-complete:

[UNIFORM EMULATION ON A PATH, for binary trees]

Instance: An undirected, connected graph G, that is a tree with each node of degree at most 3, and $n \in N^+$.

Question: Is there a uniform emulation of G on P_n?

Proof. Clearly the problem is in NP. To prove NP-completeness we will transform the BANDWIDTH problem for trees with each node of degree 3 or less to this problem. This version of BANDWIDTH is NP-complete [GJ79].

Let $G = (V_G, E_G)$ be a tree with every node degree 3 or less; and let an integer K, $o \leq K \leq |V_G|$ be given. Let $n = |V_G|$.

We will construct an undirected graph $G' = (V_1, E_1)$, that is (again) a tree with every node degree 3 or less, and an $m \in N^+$, such that G' can be uniformly emulated on P_m, if and only if G has bandwidth at most K.

We can choose $k \in N^+$ such that (k-1) is a power of 2, $2^k \geq 42kK$ and k is polynomially bounded in log n. Let $c = 2^{k+1}/(k+1)$. Notice such a k can be found (in polynomial time in log n) and c is polynomially bounded in n. c will be the computation factor of the (possible) resulting uniform emulation. Let $m = 4(k+1) + 6k(n-1) + \frac{3}{4}(k+1)n$.

We now describe the construction of G'. The main idea is similar to that in 7.4.2.1. G' consists of the following parts:

a) To replace the blocks of 7.4.2.1. we take 2 perfect binary trees of depth k (i.e., with 2^k leaves, and 2^k-1 internal nodes each). The two roots of the trees are connected by an edge (see fig. 7.4.3.1.). We take two of these "tree blocks". They fulfill the same role as the blocks in the proof of theorem 7.4.2.1.

b) take a path with length $6k(n-1)+\frac{3}{4}(k+1)n$ between a leaf of one of the tree blocks and a leaf of the other tree block.

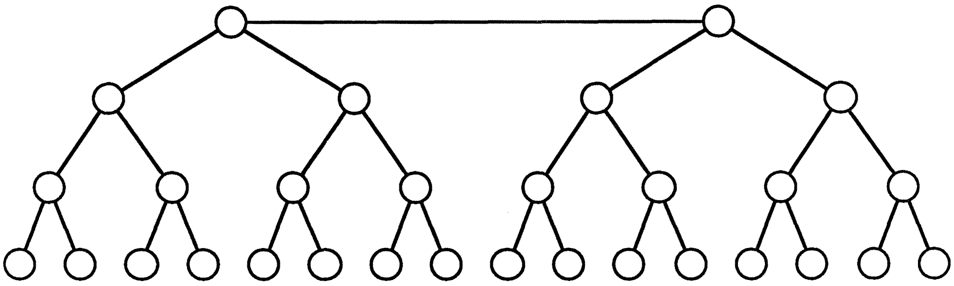


fig. 7.4.3.1.

c) In the proof of theorem 7.4.2.1 we managed that every other node had a lot of nodes of type c) mapped to it; thereby creating "slots" and "mountains". Here "slots" and "mountains" will use more consecutive nodes on P_m.

We take a perfect binary tree of depth k and remove (K+4)k leaves. A copy of this tree is connected to the j'th node of the path of b), for each $j \in \{\frac{3}{4}(k+1)i + 6k(i-1) + \alpha \mid 1 \le i \le n-1,\ \alpha \in \{k, k+1, 3k, 3k+1, 5k, 5k+1\}\}$.

In this way we create n "slots" of $\frac{3}{4}(k+1)$ consecutive nodes each on P_m and n−1 "mountains" of 6k nodes each on P_m.

d) The nodes x of V_G are represented by (k−1)-deep perfect binary trees T_x.

e) For each x,y $(x,y) \in E_G$, we connect a leaf of T_x and a leaf of T_y with a path of length $\frac{3}{4}(k+1)(K+1)+6kK$.

f) Connect an arbitrary leaf of an arbitrary T_x to the first node of the path of b), with a path of length $\frac{3}{4}(k+1)n + 6k(n-1)$.

g) Add to the last node of the path of b) a path with just enough nodes to total the total number of nodes to c.m.

Let G' be the graph so obtained. G' is a tree, and without difficulty G' can be chosen, such that no node has degree more than 3.

<u>Claim 7.4.3.1.1</u>. G' can be uniformly emulated on P_m, if and only if G

has bandwidth at most K.

Proof. We will only stress the difference with the proof of theorem 7.4.2.1. Again we will say that a node p in P_m has α free places left if we have designated $c-\alpha$ nodes of G' so far to be mapped on p.

First let a linear ordering $f:V_G \rightarrow V_n$ with bandwidth $\leq$ K be given. We will show there is a uniform emulation g of G' on P_m.

a) The constructions of a) are mapped on the first 2k+2 and the last 2k+2 nodes of P_m. We will only see how the first construction must be mapped on the first 2k+2 nodes of P_m (the other construction must be mapped in a similar way on the last 2k+2 nodes).

One of the trees of the construction is connected to the path of (b), the other tree not. We map c leaves of the tree not connected to the path to o, and $\frac{1}{2}c$ leaves of this tree on each of the nodes 1,2,...,k-1. In this way we have mapped every leaf of this tree. We let nodes with similar ancestors be mapped as much as possible on the same node. Then we can map the parents of the nodes, mapped upon i on i+1, for each i $\in$ {o,...,k-1}. The rest of this tree we map on k. When we would map the other half of the tree in the same manner on {k+1,...,2k+1}, then every node in {o,...,2k+1} would have mapped c nodes upon it, except k and k+1, which would have mapped c-1 nodes upon it. We can now alter this mapping slightly (we move some nodes one place towards the begin on P_m), such that every node in {o,...,2k} has c nodes mapped upon it, and 2k+1 has c-2 nodes mapped upon it. One has to take care that the leaf, connected to the path of (b) is mapped to 2k+1.

b) We now can map the nodes of the path (b) to the nodes 2k+2,..., $2k+1+6k(n-1)+\frac{3}{4}(k+1)n$, (i.e. the j'th node of the path is mapped to 2k+1+j.)

c) We can map the trees of (c) in a manner, similar to the mapping of the trees in (a), in such a way that

- a tree connected to the j'th node of the path (b), where $j \in \{\frac{3}{4}(k+1)i + 6k(i-1)+ \alpha \mid 1 \leq i \leq n-1 \ ; \ \alpha \in \{k,3k,5k\}\}$ is mapped to the nodes {2(k+1)+j-k , ..., 2(k+1)+j-1}.
- a tree connected to the j'th node of the path (b), where j $\in$

$\{\frac{3}{4}(k+1)i + 6k(i-1) + \alpha \mid 1 \le i \le n-1;\ \alpha \in \{k+1,3k+1,5k+1\}\}$ is mapped to the nodes $\{2(k+1)+j-1,\dots,2(k+1)+j+k-2\}$

- every node in H has at most $c-(K+4)$ such nodes mapped upon it.

d) Now notice we have n "slots" $S_i = \{\frac{3}{4}(k+1)i + 6k(i-1) + \alpha-1 \mid 1 \le \alpha \le \frac{3}{4}(k+1)\}$ where $i \in \{o,\dots,n-1\}$. Each node in these slots has c-1 free places. We use these slots to map the T_x: We map the nodes of T_x in the slot $S_{f(x)}$. Note that $|T_x| = 2^k-1 = \frac{1}{2}(k+1)c-1$, so we have in a slot in total $\frac{3}{4}(k+1)c - (\frac{1}{2}(k+1)c-1) - \frac{3}{4}(k+1) = \frac{1}{4}(k+1)(c-3)+1$ free places left. We have to take care that in each node in the slot at least k+6 free places are left. This can be done, in a manner similar to (a).

e) Now notice if $(x,y) \in E_G$ then the trees T_x and T_y are at most K slots apart, so the leaves of these trees have distance at most $\frac{3}{4}(k+1)(K+1)+6kK$. So we can lay out the paths of (e). If nodes $f(x),f(y)$ have distance less than K then we must fold the path between T_x and T_y. This can be done in the space left in one of the slots $S_{f(x)-1}$, $S_{f(y)-1}$, such that in every node in a slot at least 3 free places are left. We need $\frac{3}{4}(k+1)$ places in a slot for the path of (b), 2^k-1 places for the tree of (c), at most $\frac{3}{4}(k+1)K$ places for the paths that connect node-trees T_x placed in other slots, at most $3(\frac{3}{4}(k+1)(K-1)+6k(k-1))$ places for the folding of the paths to the node-tree T_x in the slot and $3\frac{3}{4}(k+1)$ places for the nodes of (f) and (g). Now $\frac{3}{4}(k+1) + 2^k-1 + \frac{3}{4}(k+1)K + 3(\frac{3}{4}(k+1)(K+1))+6k(K-1)) + 3\cdot\frac{3}{4}k \le 2^k+21(k+1)K \le \frac{3}{4}(k+1)c$.

The nodes of f) and g) can be mapped in a manner, similar to 7.4.2.1.1.

Now let g be a uniform emulation of G′ on P_m. We will construct a linear ordering $f : V_G \to V_n$ with bandwidth $\le K$. First look at the images of the constructions of (a). If for a construction the two roots of the two trees are mapped upon the same node, then we derive a contradiction: every node of the construction has distance $\le k$ to one of the roots and there are $2.2^{k+1}-2$ such nodes. This means that $2.2^{k+1}-2$ nodes are mapped upon the image of the roots and the 2k nodes with distance $\le k$ in P_m. This contradicts uniformity. So the roots of the trees are mapped upon two neighboring nodes and the $2.2^{k+1}-4 = 2(k+1)c-4$ other nodes are mapped upon the 2(k+1)nodes with distance at most k to one of the images of the roots. This means that in this 2(k+1) nodes in P_m only

2 free places (total) are left, so these constructions function in the same way as the blocks in 7.4.2.1.1. they have to be mapped to the ends of P_m; the path (b) is stretched out along P_m: the i'th node of the path must be mapped upon $i+2(k+1)-1$.

Now look at the images of the roots of T_x, for all $x \in V_G$. If there is such a root r_x, with $g(r_x) \in \{2(k+1)+\frac{3}{4}(k+1)i + 6k(i-1) + \alpha | o \leq i \leq n-1,\ k-1 \leq \alpha \leq 5k\}$ (that is, $g(r_x)$ is more than k-1 nodes of "a slot" away), then we can reach a contradiction: on the 2k-1 nodes $g(r_x)-k+1,\ldots,g(r_x)+k-1$ are mapped not only the 2^k-1 nodes of T_x, but also at least $(2k-1)(c-K-4)-2k(K+4)-2c$ nodes of the trees of (c). Because $2^k-1+(2k-1)(c-K-4)-2k(K+4)-2c > (2k-1)c$, a contradiction with uniformity arises. (Use $2^k \geq 42$ kK and $c= 2^{k+1}/(k+1)$.) In a similar manner one can prove that in an interval $2(k+1)+\frac{3}{4}(k+1)i+6ki-k-1,\ldots,$ $2(k+1)+\frac{3}{4}(k+1)(i+1)+6ki+k-1$ (that is a slot and k nodes at each side of a slot) nodes of at most one tree T_x are mapped. So to trees T_x we can assign a (unique) slot, in which, or near which it is placed. Now define $f(x)=i$, if the root of T_x, r_x is mapped in $[2(k+1) + \frac{3}{4}(k+1)i + 6ki-k-1, 2(k+1)+\frac{3}{4}(k+1)(i+1) + 6ki +k-1]$, i.e. T_x is placed in the i+1'th slot. If $(x,x') \in E_G$, then between T_x and $T_{x'}$ there is a path with length $\frac{3}{4}(k+1)(K+1) + 6kK$: if T_x and $T_{x'}$ are placed more than K slots away, then the minimum distance between a node of T_x and a node of $T_{x'}$ is $6\ k(K+1)+ \frac{3}{4}(k+1)K -4K > \frac{3}{4}(k+1)(K+1) + 6kK$. (Use that $T_{x'}$, $T_{x'}$ have a depth k-1). Contradiction. So $(x,x')\in E_G \Rightarrow | f(x)-f(x') | \leq K$, i.e. f is a linear ordering with bandwidth $\leq$ K. □

Again one can adept this proof for the case of uniform emulation on the ring.

Corollary 7.4.3.2. The following problem is NP-complete:

[UNIFORM EMULATION ON A RING for binary trees]

Instance: An undirected connected graph G, that is a tree with each node of degree at most 3 and an $n \in N^+$.

Question: Is there a uniform emulation of G on R_n?

7.4.4. NP-completeness for directed graphs. The techniques used in sections 7.4.2. and 7.4.3. can be used to prove NP-completeness results for the directed versions of the problems there addressed. The question whether uniform emulations of certain graphs exist on a directed path is not very interesting, because the directed path is not strongly connected. For completeness we mention a result for uniform emulation on the directed path too.

We will use the DIRECTED BANDWIDTH problem, i.e., the problem to determine whether a connected (but in general not strongly connected) graph has a bandwidth of K or less. This problem is NP-complete [GJ79]. We remark that it is not difficult to prove the DIRECTED CYCLIC BANDWIDTH problem (for connected graphs) NP-complete too (reduction to DIRECTED BANDWIDTH).

Theorem 7.4.4.1. The following problem is NP-complete:

[UNIFORM EMULATION ON A DIRECTED RING]

Instance: A directed, strongly connected graph G and an $n \in N^+$.

Question: Is there a uniform emulation of G on R_n?

Proof. We use similar techniques as in section 3 and 4. Clearly the problem is in NP. To prove NP-completeness we transform DIRECTED BANDWIDTH to this problem.

Let a directed graph $G = (V_G, E_G)$, and a $K \leq |V_G|$, $K \in N^+$ be given. We will construct a graph G', that is strongly connected and G' can be uniformly emulated on $R_{|V_G|+1}$, if and only if G has bandwidth at most K. (We suppose $|V_G| \geq 3$.)

Let $n=|V_G|$. Let $c= K(K-1)+2n+4$. We let G' consist of the following parts:

a) one 'block' B: a clique with c nodes (nodes $v_1,\ldots,v_c$, edges (v_i,v_j) for all $1 \leq i \neq j \leq c$)

b) nodes x of V_G are represented by cliques C_x of $\frac{1}{2}$ c nodes: (nodes $w_{x,i}$, $x \in V_G$, $1 \leq i \leq \frac{1}{2}c$, $(w_{x,i}, w_{x,j}) \in E_G$ for all $i \neq j$, $x \in V_G$)

c) a path of length n from v_1 ($\in$B) to $w_{x,1}$ for every $x \in V_G$ is added, (so there are n such paths).

d) also a path of length n from $w_{x,1}$ to v_1 ($\in$B) for every $x \in V_G$ is

added. (The paths of c) and d) make G′ strongly connected.)
e) if $(x,y) \in E$, then add a path with K-1 nodes from $w_{x,1}$ to $w_{y,1}$.
f) Let the total number of nodes used so far in a-e) be m. Add a path with $c(n+1)-m$ nodes, that goes from v_1 to v_1.

Claim 7.4.4.1.1. G has directed bandwidth of at most K, if and only if G′ can be uniformly emulated on $\vec{R}_{n+1}$.

Proof. Suppose f is a uniform emulation of G′ on $\vec{R}_{n+1}$. Now notice that if $(v_1,v_2) \in E_{G'}$ and $(v_2,v_1) \in E_{G'}$ then $f(v_1)=f(v_2)$. This makes B again an 'unpassable object' for paths, and forces the cliques to be mapped upon the same node. Because on every node at least one node of the path of (f) is mapped, there cannot be two different C_x and $C_{x'}$ that are mapped upon the same node.

Now let $f(B) = \alpha$. Then define $g: V_G \to V_n$ as follows: $g(x) = (f(w_{x,1})-\alpha)$ mod n. Notice g is a linear ordering of V_G and if $(x,y) \in E_G$ then there is a path with length K-1 from $w_{x,1}$ to $w_{y,1}$, that cannot pass over f(B) $= \alpha$, so $g(y)-g(x) \leq K-1$ and $g(y) \geq g(x)$: G has bandwidth $\leq$ K.

The 'only if' part can be easily provided, in a manner similar to 3.1.1. □

Finally note G′ can be constructed in time polynomial in $|V_G|$. Hence the stated problem is NP-complete. □

With a small refinement of the techniques we can get a slightly stronger result. We first need:

Lemma 7.4.4.2. For every $n \in N^+$, $n \geq 2$ and every $m \in N^+$, $m \geq 5$, there exists a directed graph $G_s=(V_s,E_s)$, with the following properties:
- G_s is strongly connected
- $|V_s|=n$
- each node of G_s is involved in at most 3 edges
- at least 2 nodes of G_s are involved in exactly 2 edges
- For every graph G, of which G_s is a subgraph, and every emulation f of

G on R_m, f maps every node of G_s on the same node of R_m, i.e. $|f(V_s)|=1$.

Proof. We use induction to n. For n=2, 3, 4 and 5 see fig. 7.4.4.1.

For $n \geq 6$ let A be a structure with $\lfloor \frac{n}{2} \rfloor - 1$ nodes and let B be a structure with $\lceil \frac{n}{2} \rceil - 1$ nodes, as defined by the lemma. Let $v_{A,1}$, $v_{A,2}$, $v_{B,1}$, $v_{B,2}$ the nodes with degree 2. Then the construction of fig. 7.4.4.2. satisfies the desired properties. □

Theorem 7.4.4.3. The following problem is NP-complete:

Instance: A directed, strongly connected graph $G=(V_G,E_G)$ with each node of G involved in at most 3 edges, and an $n \in N^+$.

Question: Is there a uniform emulation of G on $\vec{R}_n$?

Proof. Clearly the problem is in NP. To prove NP-completeness we will again use a transformation from DIRECTED BANDWIDTH. Let $G=(V_G,E_G)$ be a directed graph. We will again give a construction of a graph G', such that G' can be uniformly emulated on $\vec{R}_{n+3}$, if and only if Dir Bandwidth (G) $\leq$ K. Let n= $|V_G|$ and let c= K(K-1)+2n+4. We will further only stress the differences with the construction of theorem 7.4.4.1. The block B is replaced by

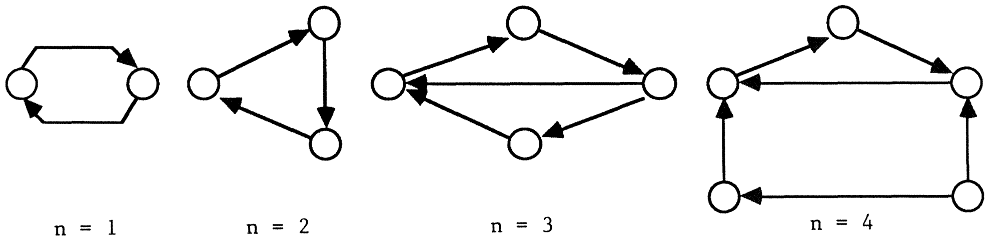


fig. 7.4.4.1.

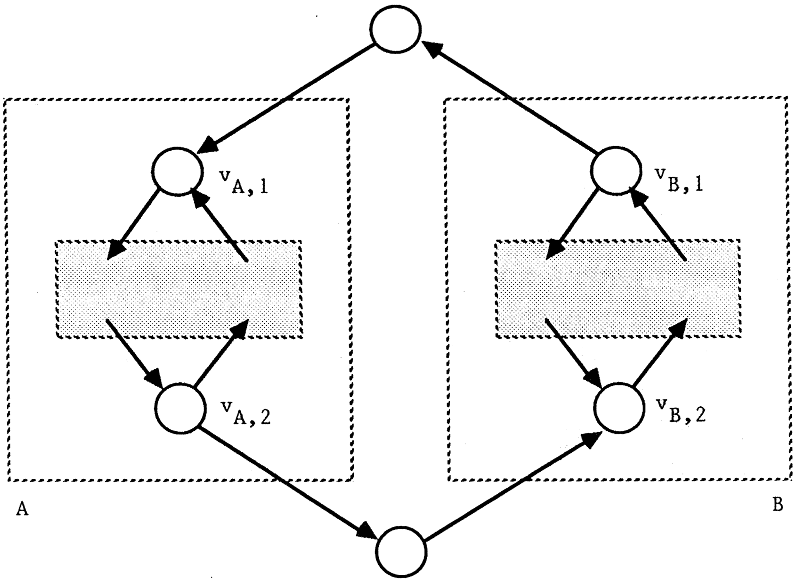


fig. 7.4.4.2.

(a1) a cycle with c nodes, with an edge from this cycle to

(a2) the structure as described in lemma 7.4.4.2., with c nodes, with an edge to

(a3) another cycle with c nodes.

Every node of (a2) must be mapped to the same node x_2 in R_{n+3}, and therefore also every node of (a1), resp. every node of (a2) must be mapped on the same node α_1, resp. α_3. The cliques C_x are replaced by cycles of $\frac{1}{2}c$ nodes: these cycles must also be mapped on the same node $\tilde{g}(x)$, because the images of (a1), (a2) and (a3) cannot be passed. The paths of (c), (a), (e), and (f) can be added in such a manner, that the relevant connections are made, but no node becomes involved in more than 3 edges. The proof that a resulting graph G′ can be uniformly emulated on $\vec{R}_{n+3}$ if and only if G has directed bandwidth K or less, is similar to 7.4.4.1. □

With a similar proof one can show:

Proposition 7.4.4.4. The following problem is NP-complete:

Instance: A directed, connected graph $G=(V,E)$, with each node involved in at most 3 edges, and an $n \in N^{+}$.

Question: Is there a uniform emulation of G on P_n?

7.4.5. Polynomial time algorithms for fixed computation factors. In the preceding sections 7.4.2. - 7.4.4. we needed for the NP-completeness proofs that the computation factor $c(=|V_G|/n)$ could grow with the size of the problem. If we fix the computation factor, then the problems become polynomial time decidable. In 1980 Saxe [Sa80] has shown, that for each constant K there exists an algorithm that uses $O(f(K)n^{K+1})$ time to determine whether a given graph $G=(V,E)$ with $|V|=n$ has bandwidth K or less. Monien and Sudborough (see [GS84]) improved the running time of the algorithm to $O(f(K)n^{K})$. (We will further let K, resp. c be constant, and do not mention factors, like $f(K)$.)

For each constant c, we will give an algorithm that determines in $O(n^{2c-1})$ time whether a connected graph $G=(V,E)$ with $|V|=n$ can be uniformly emulated on $P_{n/c}$, and a similar algorithm, that uses $O(n^{3c-1})$ time, and determines whether a connected G can be uniformly emulated on $R_{n/c}$. The algorithms are a rather straightforward modification of Saxe's algorithms.

First note that if there is a node v in G with degree 3c or more then G cannot possibly be emulated on P_n (or R_n): every node adjacent to v, and v itself must be mapped upon $f(v)-1$, $f(v)$ or $f(v)+1$. So we may assume without loss of generality that every node of G has degree $3c-1$ or less. We will now introduce the notion of a partial uniform emulation:

Definition. A partial uniform emulation of $G=(V,E)$ on $P_{n/c}$ is a function f from some subset V' of V onto $\{o,\ldots,M-1\}$, for some M such that $1 \le M \le n/c$, such that:

(i) $\forall i \in \{o,\ldots,M-1\}$ $|f^{-1}(i)|=c$.

(ii) If $u,v \in V'$ and $(u,v)\in E$ then $f(u)=f(v)$ or $(f(u),f(v))\in E_n^{p}$ (i.e. $|f(u)-f(v)|\le 1$).

(iii) If $u \in V'$ and $f(u) \le M-2$ and $(u,v)\in E$ then $v \in V'$.

We say that f is feasible iff it can be extended to a (total) uniform emulation of G on $P_{n/c}$. Notice that for every mapping f of some subset $V' \subseteq V$ onto $\{o\ldots M-1\}$, condition (i), (ii), and (iii) are necessary to be feasible.

Definition. For a set $S \subseteq V$ let $Adj(S)=\{w \in V|$ w is adjacent to a node $v \in S$ and $w \notin S\}$.

Definition. The active region of a partial uniform emulation (on $P_{n/c}$) f is the set $f^{-1}(M-1)$, together with the set $Adj(f^{-1}(M-1))\backslash f^{-1}(M-2)$.

(Note: the active region of a partial uniform emulation is the set of nodes mapped upon the last-so-far used node and the nodes that have to be mapped upon the next node.)

Theorem 7.4.5.1. Let f and g be two partial uniform emulations of G on $P_{n/c}$ with identical active regions. Then

i) f and g have identical domains

and ii) f is feasible iff g is feasible.

Proof. i) Let the active region of f (and g) be (S,A). G is connected, so a node v is in the domain of f, or of g, if and only if it is path connected to a node in S by a path, that does not use any node in A. So the domains of f and g are the same.

ii) Note that any mapping of the remaining vertices to $\{m,\ldots,n/c\}$, which extends either f or g to a uniform emulation must also extend the other to a uniform emulation. □

Now we say that two partial uniform emulations are equivalent, if they have identical active regions. Note that the number of different active regions is bounded: each node has a degree of 3c-1, so $|Adj(c)| \leq (3c-1)c$, so there are at most $2^{(3c-1)c}\cdot(\frac{n}{c}) = O(n^c)$ different active regions, or equivalence classes. The algorithm is essentially a breadth-first search over the space of all equivalence classes of partial uniform emulations. The algorithm uses the following data structures:

1. A queue Q, whose elements are active regions
2. An array A, which contains one element with two fields for each active region. For each active region r A[r].examined is a boolean which will be set to true when r is examined for the first time, and A[r].unplaced is a list of the nodes, not in the domain of each partial uniform emulation with active region r.

We will now give the algorithm:

Algorithm P

```
1. Set all A[r].examined FALSE.
2. Let Q be the empty queue.
3. (Set all the active regions of partial uniform emulations with $M=1$ on
   the queue).
   For each set $S \subseteq V$ with $|S|=c$, perform the following steps:
   a. Let s be the active region $s = (S, Adj(S))$.
   b. Set A[s].examined TRUE.
   c. Set A[s].unplaced to be $V \setminus S$.
   d. Insert s at the end of Q.
4. Extract an active region r from the head of Q.
5. Suppose $r=(R,A)$.
   If $|A| > c$ then go to step 7. (r cannot be extended to a uniform emu-
   lation, so does not have to be considered.)
6. For each set $S \subseteq$ A[r].unplaced, with $A \cap S = \emptyset$, and $|A| + |S| =$
   $|A \cup S| = c$, perform the following steps:
   a. Compute $B= Adj(A \cup S) \setminus R$.
      Let s be the active region $s = (A \cup S,B)$.
   b. If A[s].examined = FALSE then perform the following steps:
      i) Set A[s].examined on TRUE.
      ii) Let A[s].unplaced be A[r].unplaced$\setminus(S \cup A)$.
      iii) If A[s].unplaced is the empty set, then halt:
      there exists a uniform emulation of G on $R_{n/c}$.
      iv) Insert s at the end of Q.
```

7. If Q is empty, then halt: there does not exist a uniform emulation of G on $R_{n/c}$. Otherwise go to step 4.

In step 3 we place every active region of partial uniform emulations with M=1 (so with the first c nodes mapped) on the stack. In step 6 the partial uniform emulation(s), whose active region is (are) considered is (are) extended by mapping c nodes on the first node in $P_{n/c}$ that still has no nodes mapped upon. Note that for every extension $\bar{f}$ of a partial uniform emulation with active region $(f^{-1}(M-1), \text{Adj}(f^{-1}(M-1)\backslash f^{-1}(M-2))$ it must hold that $\bar{f}(\text{Adj}(f^{-1}(M-1)^{-1}(M-2)) = \{M\}$.

The space required by this algorithm is $O(n^{c+1})$. Now estimate the (worst-case) running time of the algorithm. Note that we first have to test whether there exists a node with degree at least 3c. This can be done in time O(n). Step 1 costs $O(n^c)$ time, step 2 O(1), step 3 costs $O(n^{c+1})$ time. For each active region r, step 4-7 are performed at most once. So the cost for step 4,5 and 7 are at most $O(n^c)$. To calculate the costs of step 6, note that A must contain at least one node (G is connected), so step 6a and the test of step 6 b are performed at most $O(n^{2c-1})$ times. For every active region, steps 6b(i)-6b(iv) are performed at most once. The amount of computation needed for one execution of steps 6b(i)-6b(iv) is O(n) so the total cost of steps 6b(i)-6b(iv) becomes $O(n^{c+1})$. So the total time the algorithm costs is $O(n^{2c-1})$ (if $c \geq 2$).

Theorem 7.4.5.2. Let $c \in N^+$ be a constant. There exists an $O(n^{2c-1})$ time and $O(n^{c+1})$ space algorithm, which determines whether a connected graph $G=(V,E)$ with $|V|=n$ and $c|n$ can be uniformly emulated on $P_{n/c}$.

Proof. If c=1 then test in linear time whether G is isomorphic to $P_{n/c}$. If $c \geq 2$ then test whether there exists a node with degree $\geq 3c$. If such a node does not exist, then use Algorithm P. □

Given a constant $c \in N^+$, this algorithm can be adepted to testing whether a connected graph $G=(V,E)$ with $|V|=n$ and $c|n$, can be uniformly emulated on $R_{n/c}$.

Now we let a partial uniform emulation of $G=(V,E)$ on $R_{n/c}$ be a function f from some subset $V' \subseteq V$ onto $\{o,\dots,M-1\}$, for some M, such that $1 \leq M \leq n/c$, such that

(i) $\forall\ i \in \{o,\dots,M-1\}\ |f^{-1}(i)|=c$.

(ii) If $u,v \in V'$ and $(u,v)\in E$ then $f(u)=f(v)$ or $(f(u),f(v)) \in E^R_n$.

(iii) If $u \in V'$ and $1\leq f(u) \leq M-2$ and $(u,v) \in E$ then $v \in V$.

Again f is <u>feasible</u> iff it can be extended to a (total) uniform emulation of G on $P_{n/c}$. Notice that for every mapping f of some subsets $V' \subseteq V$ onto $\{o\dots M-1)$, conditions (i),(ii) and (iii) are necessary to let f be feasible.

The active region of a partial uniform emulation f is now the 4-tuple $(f^{-1}(M-1), Adj(f^{-1}(M-1))\backslash f^{-1}(M-2), f^{-1}(o), Adj(f^{-1}(o))\backslash f^{-1}(1))$. (We will not consider partial uniform emulations for $M<2$.)

<u>Theorem 7.4.5.3</u>. Let f and g be two partial uniform emulations of G on $R_{n/c}$, with identical active regions. Then

i) f and g have identical domains.

ii) f is feasible iff g is feasible.

<u>Proof</u>. i) Let (S,A,T,B) be the active region of f and g. A node v is in the domain of f (or of g) if and only if it is path connected to a node in S or to a node in T by a path not using any nodes in A and B. So f and g have identical domains.

ii) similar to 7.4.5.1. □

First we test whether G has a node with degree 3c or more. Suppose such a node does not exist.

Let v* be an arbitrary node in V. For symmetry reasons we may suppose that $f(v^*)=o$. So we only have to consider active regions (S,A,T,B) with $v^* \in T$. The number of different active regions with $v^* \in T$ is $O(n^{2c-1})$. The algorithms uses a queue Q and array A as before: there is an entry in A for each active region with $v^* \in T$. The algorithm now becomes:

Algorithm R

```
1. Set all A[r].examined FALSE.
2. Let Q be the empty queue.
3. For each pair of sets $S,\underline{S} \subseteq V$ with
   $|S|=|\underline{S}|=c$ and $v^* \in \underline{S}$ and $S \cap \underline{S} = \emptyset$, perform the following steps:
   a. Let S be the active region $(S, Adj(S)\backslash\underline{S}, \underline{S}, Adj(\underline{S})\backslash S)$.
   b. Set A[s].examined TRUE.
   c. Set A[s].unplaced to be $V\backslash(S \cup \underline{S})$.
   d. Insert s at the end of Q.
4. Extract an active region r from the head of Q.
5. Suppose r= (R,A,S,B).
   If $|A| > c$ or $|B| > c$ then go to step 7.
6. For each set $T \subseteq$ A[r].unplaced with $A \cap T = \emptyset$ and $|A|+|T|=$
   $|A \cup T|= c$ perform the following steps:
   a. Compute $C = Adj(A \cup T)\backslash R$.
      Let s be the active region $(A \cup T, C, S, B)$.
   b. If A[s].examined= FALSE then perform the following steps:
      i) Set A[s].examined on TRUE.
      ii) Let A[s].unplaced be A[r].unplaced $\backslash(A \cup T)$.
      iii) If A[s].unplaced is the empty set,
            then if $B \subseteq A \cup T$ then halt: there exist a uniform emulation on $R_{n/c}$.
            else $B \not\subseteq A \cup T$ then go to step 7. (First note that there
                 exists exactly one $T \subseteq A[r]$ fulfilling the requests
                 (|A[r].unplaced|=c); and this T yields a mapping that
                 is not an emulation: "the ends of the ring do not fit
                 together").
      iv) Insert s at the end of Q.
7. If Q is empty then halt: there does not exist a uniform emulation of
   Q on $R_{n/c}$. Otherwise go to step 4.
```

The algorithm uses $O(n^{2c})$ space (for each active region (S,A,T,B) with $v^* \in T$ $O(n)$ space is used). To estimate the time, note that we can no longer suppose that for an active region (S,A,T,B), $A \neq \emptyset$. So step 6a and the test of step 6b cost $O(n^{3c-1})$ time. If $c \geq 2$ then the cost of the other steps is smaller, so the time used by the algorithm is $O(n^{3c-1})$.

Theorem 7.4.5.4. Let $c \in N^+$ be a constant. There exists an $O(n^{3c-1})$ time and $O(n^{c+1})$ space algorithm which determines whether a connected graph $G=(V,E)$ with $|V|=n$ and $c|n$ can be uniformly emulated on $R_{n/c}$.

For the directed case the same results hold.

7.4.6. Disconnected graphs. It is interesting to see, that if we drop the requirement that G is connected, then the problem to determine whether $G=(V,E)$ can be uniformly emulated in $P_{n/c}$ (or $R_{n/c}$) ($|V|=n$, $c|n$) becomes NP-complete, even for fixed computation factors c.

Theorem 7.4.6.1. For every $c \in N^+$, $c \geq 4$, the following problem is NP-complete:

Instance: An undirected graph $G=(V,E)$, with $n=|V|$, $c|n$.

Question: Is there a uniform emulation of G on $P_{n/c}$?

Proof. We use a technique which is similar to a technique used in [LVW84]. Clearly the problem is in NP. To prove NP-completeness we transform 3-PARTITION to it. 3-PARTITION is the following problem:

[3-PARTITION]

Instance: A set A of 3m elements, a bound $B \in N^+$, and a size $s(a) \in N^+$ for each $a \in A$, such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a)=mB$.

Question: Can A be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that for each i, $1 \leq i \leq m$ $\sum_{a \in A_i} s(a)=B$?

3-PARTITION is NP-complete in the strong sense [GJ79]. (Note that necessarily $|A_i|=3$ for each i, $1 \leq i \leq m$.)

Let an instance of 3-PARTITION be given, i.e. a set A, $|A|=3m$, $B \in N^+$, and a size $s(a) \in N^+$ for each $a \in A$, with $B/4 < s(a) < B/2$, and $\sum_{a \in A} s(a)=mB$.

Let $W \subseteq N$ be the set $\{0,1,B+2,2B+3,3B+4,\dots,mB+B+1,mB+B+2\}$. We will create a graph G^o, that when emulated on P_{mB+B+3} such that each node in P_{mB+B+3} has at most c node mapped upon it, forces that each node in W has c nodes mapped upon it, and each node in $\{0,\dots,mB+B+2\}\backslash W$ has c−1 nodes mapped upon it. To this graph G^o we add for each $a \in A$ a copy of $P_{s(a)}$ (a path with length s(a)).

The resulting graph G can be uniformly emulated on P_{mB+B+3} iff there is a solution to the 3-PARTITION problem: the paths must be mapped upon the remaining free places, after G^o is mapped: there are m parts of exactly B consecutive nodes with one free place each.

Definition. $G^o=(V^o,E^o)$ is defined as follows: $V^o=\{v_{i,j} \mid o \le i \le mB+B+2 \quad 1 \le j \le c$ and $(j \ne c$ or $i \in W)\}$, $E^o = \{v_{i_1,j_1}, v_{i_2,j_2} \mid v_{i_1,j_1} \ne v_{i_2,j_2} \in V^o$ and $|i_1-i_2| \le 1$ and $(i_1 \ne i_2$ or $j_1 \ne j_2)\}$.

Lemma 7.4.6.1.1.. For every emulation f of G^o on P_{mB+B+3}, such that $|f^{-1}(i)| \le c$ for every $i \in V_{mB+B+3}$:

i) Either $f(v_{i,j})=i$ for all $v_{i,j} \in V_o$ or $f(v_{i,j})=mB+B+2-i$ for all $v_{i,j} \in V_o$ and

ii) If $i \in W$ then $|f^{-1}(i)|=c$.
If $i \in V_{mB+B+3}\backslash W$ then $|f^{-1}(i)|=c-1$.

Proof. Omitted. (For this proof it is essential that $c \ge 4$.)

Now let $G=(V,E)$ be defined by $V=V^o \cup \{(a)_i \mid a \in A \wedge o \le i \le s(a)-1\}$ and $E=E^o \cup \{((a)_i,(a)_j) \mid (a)_i,(a)_j \in V\backslash V^o \wedge |i-j| = 1\}$. Note that $|V|=|V_o|+\sum_{a \in A} s(A)=c\cdot(mB+B+3)$.

Lemma 7.4.6.1.2. G can be uniformly emulated on P_{mB+B+3} if and only if A can be partitioned in m disjoint sets $A_1,\dots,A_m$ such that for each i, $1 \le i \le m \quad \sum_{a \in A_i} s(a)=B$.

Proof. Suppose an emulation f of G on P_{mB+B+3} exists. For each i, $1 \le i \le m$, look at $R=f^{-1}(\{i-1)\cdot(B+1)+2,...,(i-(B+1)\})$. Necessarily this set exists of $(c-1)\cdot B$ nodes $\in V^o$, and B nodes $\in V\backslash V^o$. There must be 3 elements $(a^1),(a^2),(a^3)$ $\in A$ with $(a^i)_j \in R$ for $o\leq j\leq s(a^i)-1$, $1\leq i\leq 3$. Now let A_i consist of (a^1), $(a^2),(a^3)$. In this way a correct partition is obtained.

Now suppose a partition of A in $A_1...A_m$ with $\sum_{a\in A_i} s(a)=B$ $(1\leq i\leq m)$ is given. Let $f(v_{i,j})=i$. For each i, $1\leq i\leq m$ we can map the nodes in $\{(a)_j|a\in A_i,1\leq j\leq s(a)\}$ on the nodes $(i-1)(B+1)+2,...,i(B+1)$. (Each of these nodes has exactly one free place left after mapping of G^o). In this way a uniform emulation of G on P_{mB+B+3} can be obtained. □

The constraint $c\geq 4$ is essential for the proof. If $c\leq 3$ then nodes, that we want to be mapped on consecutive nodes can be mapped on the same node. (See fig. 7.4.6.1.)

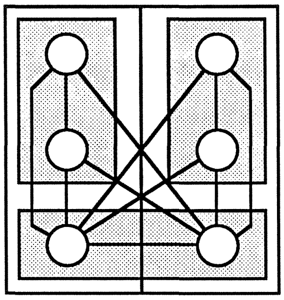


fig. 7.4.6.1.

In fact, we conjecture that there exist polynomial time algorithms for the considered problem for c=2 and c=3. (For c=1 it is trivial that a polynomial algorithm exists.) With a similar technique one can prove:

Theorem 7.4.6.2. For every $c \in N^+$, $c\geq 2$ the following problem is NP-complete:

Instance: An undirected graph $G=(V,E)$ with $|V|=n$ and $c|n$.

Question: Is there a uniform emulation of G on $R_{n/c}$?

The same results hold for the directed versions of the problems.

7.5. The complexity of finding uniform emulations on fixed graphs. In this section we show that for every fixed graph H, that is connected (or strongly connected in the case of directed graphs) but not complete, the problem to decide whether another (strongly) connected graph G can be uniformly emulated on H is NP-complete.

Definition. Let $G=(V,E)$ be an undirected, bipartite graph. We say that G contains a balanced complete bipartite subgraph (abbreviated as BCBS) of $2 * K$ nodes, if there are two disjoint subsets $V_1, V_2 \subseteq V$, such that $|V_1| = |V_2| = K$ and $u \in V_1$, $v \in V_2$ implies that $\{u,v\} \in E$.

The problem to decide, given a bipartite graph G, and a $K \in N^+$, whether G contains a BCBS with $2 * K$ nodes is NP-complete [GJ79]. We now prove the following variant of BCBS to be NP-complete.

Lemma 7.5.1. Let $n \in N^+$, $n \geq 3$. The following problem is NP-complete:

Instance: Bipartite graph $G=(V,E)$ with $n \mid |V|$, and at least one node of V is isolated (i.e. is of degree 0).

Question: Are there two disjoint subsets $V_1, V_2 \subseteq V_1$, such that $u \in V_1$, $v \in V_2$ implies that $\{u,v\} \in E$ and $|V_1|=|V_2|= \frac{1}{n} \cdot |V|$?

Proof: Clearly the problem is in NP. To prove NP-completeness we transform the BALANCED COMPLETE BIPARTITE SUBGRAPH problem to it.

Let an instance of BCBS be given. We consider three cases:

Case I: $cK>|V|$.

In this case add $cK-|V|$ extra nodes of degree 0 to G. The graph G', in this way obtained, is bipartite, has at least one node of degree 0, and contains a BCBS with $2*K = 2*\frac{1}{c} \cdot |V'|$ nodes if and only if G contains a BCBS with $2*K$ nodes.

Case II: $cK<|V|$.

We transform this case to case I. In polynomial time we find disjoint sets V_a, V_b such that $V=V_a \cup V_b$, and each edge $\in E$ goes between a node of V_a and a node of V_b. (V_a, V_b are the two "halves of the bipartition".)

Now add $|V|$ extra nodes to V_a and $|V|$ extra nodes to V_b and connect each of the extra nodes to each of the nodes in the other half: we have extra nodes $v_1^a,\ldots,v_{|V|}^a$, $v_1^b,\ldots,v_{|V|}^b$ and edges (v_i^a,v_j^b) $(1\le i,j\le|V|)$; $(v_i^a,w)\,(1\le i\le|V|,\ w\in V_b)$ and $(v_i^b,w')\,(1\le i\le|V|,w'\in V_a)$. The graph $G_1=(V_1,E_1)$ so obtained has the following properties:

- G_1 is bipartite
- G_1 contains a BCBS of $2*(K+|V|)$ nodes if and only if G contains a BCBS of $2*K$ nodes.
- $c\cdot(K+|V|) > |V'|$

So now we have an instance of case I, and handle the graph as described there.

Case III: $cK=|V|$. If G does not contain isolated nodes, add one, and we are in case II.

We finally obtain in this way a graph, that has a BCBS of $2*\frac{1}{c}|V'|$ nodes if and only if G contains a BCBS of $2*K$ nodes. Note that the constructions can be carried out in time, polynomial in the size of G. □

Lemma 7.5.2. Let V_1, V_2 be disjoint finite sets, and $G=(V_1\cup V_2,E_G)$ be an undirected bipartite graph, with edges between nodes of V_1 and V_2 only, i.e., $(v,w)\in E_G \Rightarrow (v\in V_1 \Leftrightarrow w\in V_2)$. Let $\tilde{G}=(V_1\cup V_2,\tilde{E}_G)$ be the undirected, bipartite graph with $\tilde{E}_G=\{(v,w)\mid v\in V_1 \wedge w\in V_2 \wedge (v,w)\notin E_G\}$. Then there is a uniform emulation of $\tilde{G}$ on P_3 with $f(V_1)\subseteq\{0,1\}$ and $f(V_2)\subseteq\{1,2\}$ if and only if G contains a BCBS with $2*|V|/c$ nodes.

Proof: First suppose f is a uniform emulation of $\tilde{G}$ on H. Choose $W_1 = f^{-1}(0)$ and $W_2 = f^{-1}(2)$. Now $W_1\subseteq V_1$, $W_2\subseteq V_2$ and $v\in W_1$, $w\in W_2 \Rightarrow (v,w)\notin\tilde{E}_G$, hence $(v,w)\in E_G$. So G contains a BCBS with $2*|V|/c$ nodes.

Now suppose G contains a BCBS with $2*|V|/c$ nodes, i.e., there are sets $W_1\subseteq V_1$, $W_2\subseteq V_2$ with $|W_1|=|W_2|=|V|/c$ and $v\in W_1$, $w\in W_2 \Rightarrow (v,w)\in E_G \Rightarrow (v,w)\notin\tilde{E}_G$. Let $f(W_1)=0$, $f(W_2)=2$ and $f(V_1\backslash W_1)=f(V_2\backslash W_2)=1$. It is easy to check that f is a uniform emulation of $\tilde{G}$ on P_3 and $f(V_1)\subseteq\{0,1\}$, $f(V_2)\subseteq\{1,2\}$. □

Lemma 7.5.3. The following problem is NP-complete:

Instance: Disjoint finite sets V_1, V_2, undirected graph $G = (V_1 \cup V_2, E_G)$

Question: Is there a uniform emulation f of G on P_3 with $f(V_1) \subseteq \{0,1\}$, $f(V_2) \subseteq \{1,2\}$?

Proof: Use lemma 7.5.1. and 7.5.2. □

Lemma 7.5.4. Let $H= (V_H,E_H)$ be a directed graph, with $V=\{0,1,2\}$ and $(0,1) \in E_H$, $(1,2) \in E_H$, $(0,2) \notin E_H$. (H is one of the 8 graphs, given in fig. 7.5.1.) The following problem is NP-complete:

Instance: Disjoint finite sets V_1, V_2, directed graph $G= (V_1 \cup V_2,E_G)$

Question: Is there a uniform emulation f of G on H, with $f(V_1) \subseteq \{0,1\}$, $f(V_2) \subseteq \{1,2\}$?

Proof: Similar to the undirected case. □

Theorem 7.5.5. Let $H= (V_H,E_H)$ be a connected undirected graph that is not complete. The following problem is NP-complete:

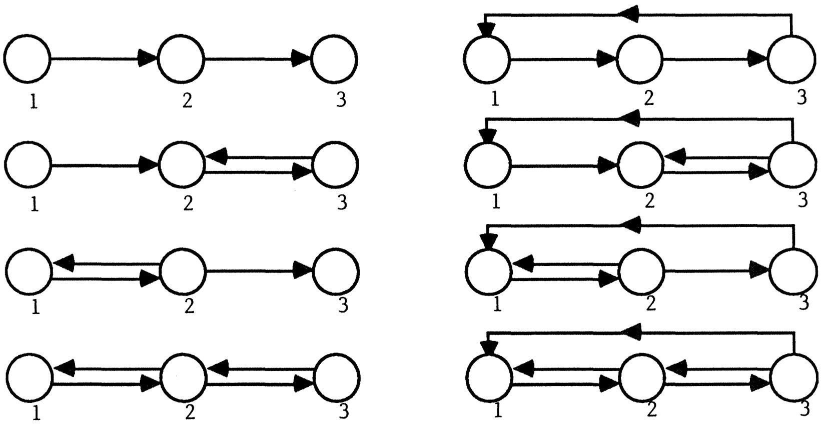


fig. 7.5.1. 8 possible choices for H in lemma 7.5.4.

Instance: A connected undirected graph $G=(V_G,E_G)$
Question: Is there a uniform emulation of G on H?

Proof: Let $H=(V_H,E_H)$ be a connected undirected graph, that is not complete. Clearly the problem is in NP. To prove NP-completeness we will transform the problem stated in theorem 7.5.3. to it.

Let disjoint finite sets V_1, V_2 and an undirected graph $G = (V_1 \cup V_2, E_G)$ be given. We will construct a connected graph $G' = (V', E')$, such that there is a uniform emulation of G' on H iff there is a uniform emulation f of G on P_3 with $f(V_1) \subseteq \{0,1\}$, $f(V_2) \subseteq \{1,2\}$. Let $c = |V_G|/3$. (c is the computation factor of the emulation of G on P_3.)

From the fact that H is not complete, it follows that there exist nodes v_0, v_1, $v_2 \in V_H$ with $(v_0,v_1) \in E_H$, $(v_1,v_2) \in E_H$ and $(v_0,v_2) \notin E_H$. (The subgraph of H, induced by $\{v_0, v_1, v_2\}$ is isomorphic to P_3.) We let G' consist of the following parts:

- 3c + 1 copies of H. From the first c copies of H, we omit the nodes v_0, v_1, v_2. We connect copies of the same and adjacent nodes.
- a copy of G.

Each copy of v_0 and v_1 is connected to each node in V_1; each copy of v_1 and v_2 is connected to each node in V_2.

Definition. Let $G' = (V', E')$ be the undirected graph, with $V' = V_G \cup \{v_{x,i} \mid x \in V_H,\ 1 \le i \le 3c+1,\ x \notin \{v_0,v_1,v_2\} \vee i \ge c+1\}$, and $E' = E_G \cup \{(v_{x,i}, v_{y,j}) \mid v_{x,i}, v_{y,j} \in V' \text{ and } (x,y) \in E_H\} \cup \{(v_{w,i},y) \mid y \in V_1,\ w \in \{v_0, v_1\}\} \cup \{(v_{w,i},y) \mid y \in V_2,\ w \in \{v_1, v_2\}\}$.

Lemma 7.5.5.1. There is a uniform emulation of G' on H if and only if there is a uniform emulation f of G on P_3 with $f(V_1) \subseteq \{0, 1\}$, $f(V_2) \subseteq \{1, 2\}$.

Proof: First suppose there is a uniform emulation g of G on P_3 with $f(V_1) \subseteq \{0, 1\}$, $f(V_2) \subseteq \{1, 2\}$. Define $f : V' \to V_H$ as follows: $f(v_{x,i})=x$ and for $y \in V_G$ $f(y)=v_{g(y)}$. It is easy to check that f is a uniform emulation of G' on H.

Now suppose there is a uniform emulation f of G′ on H. For every x $\in V_H$, let N(x) be the set consisting of x and its neighbors, i.e., $N(x)=\{x\} \cup \{y \mid (x,y) \in E_H\}$. Number the nodes in H $w_1,\ldots,w_{|V_H|}$, in order of non-increasing degree, i.e., if degree (w_i) > degree (w_j) then $i<j$.

Claim 7.5.5.1.1. $\forall i, 0\leq i\leq |V_H|$, there exists a uniform emulation f^i of G′ on H, such that $\forall j,k,l$ with $v_{w_j,k}, v_{w_j,l} \in V' \wedge 0\leq j\leq i$ $f^i(v_{w_j,k})=f^i(v_{w_j,l})$.

Proof:We use induction to i. For i=0 the claim follows immediately.

Now let f^i be given. Notice that for w_j, $1\leq j\leq i$ and $v_{w_{i+1},l} \in V'$ $f^i(v_{w_j,l})=f^i(v_{w_{i+1},l}) \Rightarrow w_j \in \{v_0,v_1,v_2\}$. Therefore if $w_{i+1} \notin \{v_0,v_1,v_2\}$, then there are at most 3c nodes $v \in \{v_{w_{i+1},l} \mid 1\leq l\leq 3c+1\}$ with $\exists j$, $1\leq j\leq i$, $f^i(v)=f^i(\{v_{w_j,k} \mid c+1\leq k\leq 3c+1\})$. If $w \in \{v_0,v_1,v_2\}$, then there are at most 2c such nodes in $\{v_{w_{i+1},l} \mid c+1\leq l\leq 3c+1\}$. Hence there exists $v_{w_{i+1},l} \in V_G$, such that there is no j, $1\leq j\leq i$, with $f^i(\{v_{w_j,k} \mid c+1\leq k\leq 3c+1\}) = f^i(v_{w_{i+1},l})$. Let such l be given and write $v = v_{w_{i+1},l}$. Look at the degree of f(v). Uniformity prevents that degree$(f(v_{x,j}))$ < degree(x), for all $v_{x,j} \in V'\backslash V_G$. Hence degree(f(v)) = degree(w_{i+1}). (We use that the degrees of $w_1,\ldots,w_{|V_H|}$ are non-increasing.)

Further notice that for every $y \in N(f^i(v))$, there is a $w \in V_G$ with $f^i(w) = y$ and $(v,w) \in E'$. Now let $f^i(v_{w_{i+1},m})\neq f^i(v)$. Every neighbor of v is a neighbor of $v_{w_{i+1},m}$. Hence $N(f^i(v_{w_{i+1},m})) \supseteq N(f^i(v))$. Choose a node y with $f^i(y)=f^i(v)$ and y not of the form $v_{w_{i+1},m}$. ($1\leq m\leq 3c+1$, if $w_{i+1} \in \{v_0,v_1,v_2\}$ then $c+1\leq m\leq 3c+1$.) For all $z \in V_G$ $(y,z) \in E_G$ implies $f^i(z) \in N(f^i(v))$. We obtain a new uniform emulation $\tilde{f}^i$ by 'exchanging' the images of $v_{w_{i+1},m}$ and y: $z \notin \{y,v_{w_{i+1},m}\} \Rightarrow \tilde{f}^i(z)=f^i(z)$, $\tilde{f}^i(y)=f^i(v_{w_{i+1},m})$ and $\tilde{f}^i(v_{w_{i+1},m})=f^i(y)$.

$\tilde{f}^i$ maps every neighbor of y upon a node in $N(f^i(v)) \subseteq N(\tilde{f}^i(y))$, and every neighbor of $v_{w_{i+1},m}$ is v or a neighbor of v, so is mapped upon a node in $N(f^i(v)) = N(\tilde{f}^i(v_{w_{i+1},m}))$. So $\tilde{f}^i$ is again a uniform emulation of G′ on H, but now $\tilde{f}^i(v_{w_{i+1},m})=\tilde{f}^i(v)$.

By repeated use of this 'image-exchanging' process one can obtain a uniform emulation f^{i+1} with $f^{i+1}(v_{w_{i+1},m})=f^{i+1}(v)$ for all $v_{w_{i+1},m} \in V' \setminus V_G$. With the induction hypothesis one proves that $\forall j,k,l$ with $v_{w_j,k}$, $v_{w_j,l} \in V'$, and $0 \le j \le i+1$ $f^{i+1}(v_{w_j,k})=f^{i+1}(v_{w_j,l})$. □

Now let $g = f_{|V_H|}$. g, restricted to the set of nodes $\{v_{3c+1} | x \in V_H\}$ can be seen as a graph isomorphism of H. Hence we have:

<u>Lemma 7.5.5.1.2.</u> There is a uniform emulation $\tilde{g}$ of G′ on H with $f(v_{x,i})=x$ for all $v_{x,i} \in V' \setminus V_G$.

Notice that if $w \in V_G$ then $\tilde{g}(w) \in \{v_0,v_1,v_2\}$, and $\tilde{g}$ maps c nodes of V_G on each of the nodes v_0, v_1, v_2. Further $w \in V_1$ implies that $\tilde{g}(w)$ must be adjacent to $\tilde{g}(\{v_{v_0,k} \mid c+1 \le k \le 3c+1\})=v_0$ and to $\tilde{g}(\{v_{v_1,k} \mid c+1 \le k \le 3c+1\})=v_1$. Hence $\tilde{g}(w) \in \{v_0,v_1\}$. Likewise $w \in V_2$ implies $\tilde{g}(w) \in \{v_1,v_2\}$. So the mapping $h : V_G \to \{0,1,2\}$, given by $h(w)=i \Leftrightarrow g(w)=v_i$ is a uniform emulation of G on P_3 with $h(V_1) \subseteq \{0, 1\}$ and $h(V_2) \subseteq \{1, 2\}$. □

Finally notice that the construction of G′ can be carried out in polynomial time in $|V_G|$. Hence the stated problem is NP-complete. □

With the following simple observation we have a complete classification of the complexity of finding uniform emulations on fixed, connected undirected graphs.

<u>Proposition 7.5.6.</u> Let $G=(V_G,E_G)$ be an undirected graph, and let K_n be the complete graph with n nodes. There is a uniform emulation of G on K_n iff $n \mid |V_G|$.

For the general case of graphs, that are not necessarily connected, we mention the following results:

Corollary 7.5.7. Let H be an undirected graph, such that at least one connected component of H is not complete. The following problem is NP-complete:

Instance: An undirected graph $G=(V_G,E_G)$
Question: Is there a uniform emulation of G on H?

Proof: Transformation from the problem of theorem 7.5.5. □

Proposition 7.5.8. Let H be a fixed undirected graph, such that each connected component of H is complete. Then there exists a polynomial time algorithm that decides whether a graph G can be uniformly emulated on H.

Proof: The problem becomes the question whether we can allocate the connected components of G to the connected components of H, such that the numbers of nodes that are allocated to components of H are proportional to its size. By exhaustive search over all allocations this is solvable in polynomial time. (We use that a graph can be separated in its connected components in linear time.) □

The general problem of deciding whether a given undirected graph G can be uniformly emulated on a given undirected graph H, with each connected component of H complete, is NP-complete. (Use a transformation from 3-PARTITION.) For directed graphs a result similar to theorem 7.5.5. can be proved.

Theorem 7.5.9. Let $H=(V_H,E_H)$ be a strongly connected directed graph, that is not complete. The following problem is NP-complete:

Instance: Strongly connected directed graph $G=(V_G,E_G)$
Question: Is there a uniform emulation of G on H?

The proof is similar to the undirected case, and uses lemma 7.5.4.

7.6. On approximation algorithms for determining minimum cost emulations.

7.6.1. Introduction. In this section we introduce the notion of computation cost, and comment on the possible existence of "good" approximation algorithms, that try to minimize the computation cost of an emulation of a network G on a network H, and run in polynomial time.

Definition. Let G,H and f be as above. The computation cost of f is $cc(f) = \max_{h \in V_H} |f^{-1}(h)|$.

Definition. Let G, H, f be as above. The emulation f is said to be (computationally) uniform iff for all h,h' ∈ V_H: $|f^{-1}(h)| = |f^{-1}(h')|$.

Proposition 7.6.1.1. . Let G,H and f be as above.
f is uniform iff $cc(f) = |V_G|/|V_H|$.

A natural extension of the UNIFORM EMULATION problem is the following problem:

[MINIMUM COST EMULATION]
Instance: Connected graphs G,H , integer $c \in N^+$.
Question: Is there an emulation f of G on H, with $cc(f) \leq c$?

We will abbreviate MINIMUM COST EMULATION as MCE.

MCE is also NP-complete: it contains UNIFORM EMULATION as a subproblem. However, it seems reasonable to look for approximation algorithms for this problem: an emulation with "relatively low cost" will enable us to simulate the given large network G on the given smaller network H "relatively efficiently".

Unfortunately there is some evidence that it will be hard to find polynomial time approximation algorithms for MCE that find "good" approximations for the computation cost of the emulation. By "good" approximations we mean e.g. approximations that give a computation cost that differs at most by a constant factor from the optimal solution.

In section 7.6.3. we show that every polynomial time approximation algorithm for MCE will give - in worst case - approximations that differ at least a factor 2 from the optimal solution (unless P=NP). In sections 7.6.4., 7.6.5. and 7.6.6 we show that the existence of a "good" polynomial time approximation algorithm for MCE will imply the existence of "good" polynomial time approximation algorithms for BANDWIDTH, CLIQUE and BALANCED COMPLETE BIPARTITE SUBGRAPH. However, for the latter three problems no polynomial time approximation algorithms are known to exist that guarantee approximations that differ for instance by a constant or even a logarithmic (in the size of the problem) factor from the optimal solution. In 1974 Johnson [J74] proved that for all of the polynomial time approximation algorithms for CLIQUE that had been suggested at the time, the worst case ratio between the approximation and the optimal value grows at least as fast as $O(n^{\varepsilon})$, where n is the problem size and $\varepsilon > o$ depends on the algorithm. Presently no algorithm that gives better ratios is known.

7.6.2. Definitions and notations. The frame work in which we present the results is taken from [GJ79, chapter 6]. An approximation problem Π consists of the following parts:

(1) a set D_{π} of instances ($D_{\pi} \neq \emptyset$).

(2) for each instance $I \in D_{\pi}$ a finite set $S_{\pi}(I)$ of candidate solutions for I ; and

(3) a function m_{π}, that assigns to each instance $I \in D_{\pi}$ and each candidate solution $\sigma \in S_{\pi(I)}$ a positive rational number $m_{\pi}(I,\sigma)$, called the solution value for σ.

We call Π a minimization (maximization) problem, if we look for (approximations of) the minimal (maximal) value of $m_{\pi}(I,\sigma)$ for a given instance $I \in D_{\pi}$ and all candidate solutions $\sigma \in S_{n}(I)$. This minimal (maximal) value is denoted by $OPT_{\pi}(I)$. (The subscript Π is usually dropped when the problem is clear from the context.)

Let A be an approximation algorithm for a minimization problem Π, (like MINIMUM COST EMULATION), and let $I \in D_{\pi}$, and A(I) be the approximation for $OPT_{\pi}(I)$, yielded by the algorithm A on input I.

Definitions. $R_A(I) = \dfrac{m_{\pi}(I,A(I))}{OPT_{\pi}(I)}$.

$R_A = \inf \{ r \geq 1 \mid \forall I \in D_{\pi} \; R_A(I) \leq r\} = \sup \{R_A(I) \mid I \in D_{\pi}\}$.

$R_A^{\infty} = \inf \{ r \geq 1 \mid \exists N \in N^{+} : \forall I \in D_{\pi} \; OPT_{\pi}(I) \geq N \Rightarrow R_A(I) \leq r\}$.

$R_{min}(\Pi) = \inf \{ r \geq 1 \mid$ there exist a polynomial time approximation algorithm A for Π with $R_A^{\infty} = r\}$.

R_A is called the "absolute performance ratio"; R_A^{∞} is called the "asymptotic performance ratio". These ratios will always satisfy $1 \leq R_A^{\infty} \leq R_A \leq \infty$. Ratios that are closer to 1 indicate a better performance. $R_{min}(\Pi)$ is called the "best achievable asymptotic performance ratio".

For a further discussion of these measures see [GJ79]. For approximation algorithms A for maximization problems define $R_A(I) = \dfrac{OPT_{\pi}(I)}{m_{\pi}(I,A(I))}$. ($R_A(I)$ denotes the ratio between the optimal value and the approximation for input I.)

7.6.3. A bound for the best achievable asymptotic performance ratio on MCE.

Theorem 7.6.3.1. If $P \neq NP$, then no polynomial time approximation algorithm A for MINIMUM COST EMULATION can satisfy $R_A^{\infty} < 2$.

Proof. Let $P \neq NP$. Suppose A is a polynomial time approximation algorithm for MINIMUM COST EMULATION with $R_A^{\infty} < 2$. Then there exist $\gamma < 2$ and $N \in N^{+}$, such that for all connected, undirected graphs G,H with $cc(G,H) \geq N$, $cc(A(G,H)) \leq \gamma . cc(G,H) < 2\, cc(G,H)$. Let such an $N \in N^{+}$ be given. (γ is not used.)

We will now give a polynomial time algorithm B_N that solves HAMILTONIAN CIRCUIT for graphs with nodes of degree 3. (This subproblem of HAMILTONIAN CIRCUIT is NP-complete [GJ79].)

Let $H = (V_H, E_H)$ be a connected, undirected graph with nodes of degree 3. The graph $H^o = (V_H^o, E_H^o)$ is obtained in the following manner: each edge $(v_o, v_1) \in E_H$ is replaced by a path with length 3: i.e. we introduce additional nodes v_{o1}, v_{1o}, and edges (v_o, v_{o1}), (v_{o1}, v_{1o}) and (v_{1o}, v_1). Furthermore to each node $v \in V_H \subseteq V_H^o$ we add two extra branches, consisting of one extra node each. An example of this transformation is given in fig. 7.6.3.1.

So $|V_H^o| = 6 \cdot |V_H|$. Let $G = (V_G, E_G)$ be the graph consisting of a cycle with $3 \cdot |V_H|$ nodes, in which to every third node on the cycle $6N-3$ branches are added, each consisting of one node (see fig. 7.6.3.2.) So $|V_G| = 6N \cdot |V_H| = N \cdot |V_H^o|$.

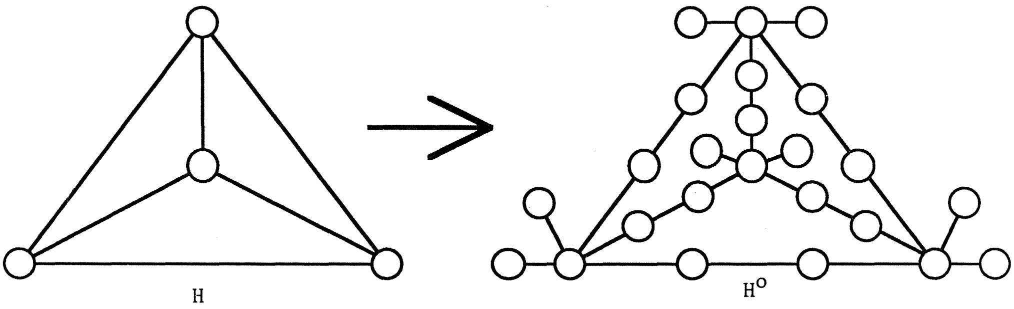


fig. 7.6.3.1.

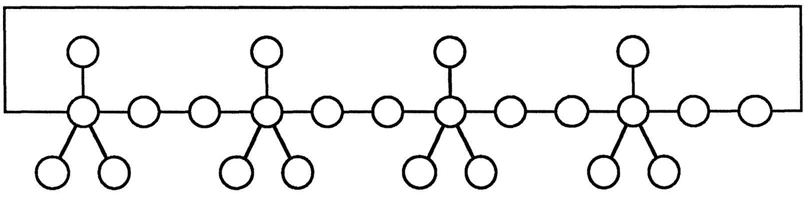


fig. 7.6.3.2. G, with N = 1, $|V_H| = 4$.

Claim 7.6.3.1.1. a. If H contains a Hamiltonian circuit, then $cc(G,H^o) = N$.

b. If H does not contain a Hamiltonian circuit, then $cc(G,H^o) \geq 2\ N$.

Proof. a). It is clear that $cc(G,H^o) \geq \frac{|V_G|}{|V_{H^o}|} = N$. Now suppose H contains a Hamiltonian circuit. We can map the successive nodes of the cycle in H with degree 6N-1 on the successive nodes $v \in V_H \subseteq V_H{}^o$, visited by the Hamiltonian circuit. The other nodes can be mapped in such a way that the resulting function f is a uniform emulation, i.e. $cc(f) = N$.

b) If an emulation f maps two nodes v_o, $v_1 \in V_G$ with $\text{degree}(v_o) = \text{degree}(v_1) = 6N-1$ on the same node $w \in V_H{}^o$, then $cc(f) \geq 2N$ (at least 12 N nodes are mapped upon 6 nodes). If f maps a node $v \in V_G$ with degree $(v) = 6N-1$ on a node $w \in V_H{}^o$ with degree $(w) = 2$, then at least 6N nodes must be mapped upon 3 nodes, so $cc(f) \geq 2N$. So if $cc(G,H^o) < 2N$, then there exists an emulation f of G on H^o, that maps nodes $v \in V_G$ with $\text{degree}(v) = 6N-1$ on nodes $W \in V_H \subseteq V_H{}^o$ on a one-to-one basis. Now let v^o, v^1, v^2, ..., $v^{|V_H|}$ be the successive nodes in G with degree 6N-1. Then $f(v^o)$, $f(v^1)$, $f(v^2)$,...,$f(v^{|V_H|})$ (seen as nodes in G) form a Hamiltonian circuit. □

From claim 7.6.3.1.1. it immediately follows that $cc(A(G,H^o)) < 2N$ if and only if H contains a Hamiltonian circuit. This gives a method to test in polynomial time whether H contains a Hamiltonian circuit. (G and H^o can be constructed in polynomial time.) This contradicts our assumption that $P \neq NP$. □

Corollary 7.6.3.2. If $P \neq NP$, then R_{min} (MCE) ≥ 2.

7.6.4. Relation of MCE with BANDWIDTH.

Definition. Let $G = (V_G, E_G)$ be a connected, undirected graph and let $f: V_G \to \{o,\ldots,|V_G|-1\}$ be a bijection. The bandwidth of f is max $\{|f(v)-f(w)| \mid (v,w) \in E_G\}$. The bandwidth of G is min ({Bandwidth (f) | f is a bijection of V_G to $\{o,\ldots,|V_G|-1\}$).

Define BANDWIDTH to be the following problem: given a graph G = (V_G, E_G), find a bijection $f : V_G \to \{o, \dots, |V_G|-1\}$ with a minimum bandwidth.

Turner [Tu86] has shown that heuristic algorithms, that fall in the class of the so-called "level algorithms", have an asymptotic performance ratio ∞. Most of the approximation algorithms, that have been proposed for the BANDWIDTH problem, (see e.g. [GPS76,WS76]) fall in this class. It is not known presently, whether there exists a polynomial time approximation algorithm A for BANDWIDTH with an absolute (or asymptotic) performance ratio that is bounded by some finite constant. (Also, Turner [Tu86] analysed the probabilistic performance of the heuristics and proposed a new approximation algorithm that produces (on the average) solutions, that are nearly optimal, but still has an asymptotic performance ration ∞.) We will see that the existence of a "good" approximation algorithm for MCE implies the existence of an approximation algorithm for BANDWIDTH that is at most "twice as bad".

Lemma 7.6.4.1. Let G=(V,E) be an undirected graph. Let $K \in N^+$, and let $n \geq \lceil |V|/K \rceil$. Then

Bandwidth (G) ≤ K

$\Rightarrow$ G can be emulated on P_n with computation cost K

$\Rightarrow$ Bandwidth (G) ≤ 2K-1.

Proof. Similar to the proof of lemma 5.2.4. □

Corollary 7.6.4.2. Suppose there exists a polynomial time approximation algorithm A for MCE with $R_A(G,H) \leq g(|V_G|, |V_H|)$ for a certain function g. Then there exists a polynomial time approximation algorithm A' for bandwidth minimization with $R_{A'}(G) \leq 2g(|V_G|, |V_G|)$.

Proof. Use algorithm A with $H = P_{|V_G|}$. □

Corollary 7.6.4.3. $R_{\min}$ (bandwidth minimization) $\leq 2\cdot R_{\min}$ (MCE).

7.6.5. Relation of MCE with CLIQUE. Define CLIQUE to be the problem, given a graph, to find the maximum subgraph all of whose points are mutually adjacent. In 1974 Johnson [J74] showed that for every polynomial time approximation algorithm for CLIQUE that had been suggested, there exists an $\varepsilon > o$, such that $R_A(G) \geq O(|V_G|^{\varepsilon})$, (for infinitely many graphs $G=(V_G,E_G)$). Presently no algorithm that gives better ratios is known.

Similar to the argument in section 4 one can show that a "good" polynomial time approximation algorithm A for MCE gives a "good" polynomial time approximation algorithm A' for CLIQUE.

Definition. The complete graph on n nodes is the graph $K_n = (V_n, E_n)$ with $V_n = \{o,\ldots,n-1\}$ and $E_n = \{(i,j) \mid i,j \in V_n \wedge i \neq j\}$.

Lemma 7.6.5.1. Let $G = (V_G,E_G)$ be an undirected graph, and $n \in N^+$.
K_n can be emulated on G with computation cost $\leq c$, if and only if G contains a clique with $\geq \lceil \frac{n}{c} \rceil$ nodes.

Proof. Suppose f emulates K_n on G with $cc(f) \leq c$. Then it is easily seen that $f(K_n)$ is a clique with at least $\lceil \frac{n}{c} \rceil$ nodes.

Suppose G contains a clique with $\lceil \frac{n}{c} \rceil$ nodes. We can map the nodes of K_n onto the nodes of this clique, such that every node of the clique has c or c-1 nodes mapped upon it. In this way an emulation f of K_n on G with $cc(f) = c$ is obtained. □

Corollary 7.6.5.2. Suppose there exists a polynomial time approximation algorithm A for MCE with $R_A(G,H) \leq g(|V_G|,|V_H|)$, for a certain function g. Then there exists a polynomial time approximation algorithm A' for CLIQUE with $R_{A'}(G) \leq g(K,|V_G|)$, for all connected graphs G of which the largest clique contains K nodes.

Proof. Let $G = (V_G,E_G)$ be given. By assumption one can calculate $A'(G) = \max \{\lceil \frac{i}{A(K_i,G)} \rceil \mid 1 \leq i \leq |V_G|\}$ in polynomial time. ($A(K_i,G)$ denotes the approximated computation cost of an emulation of K_i on G, produced by algorithm A).

Suppose the largest clique in G contains K nodes. $A'(G)$ is the desired approximation of K: for all i, $cc(K_i,G)= \lceil \frac{i}{K} \rceil$. So $A(K_i,G) \geq \lceil \frac{i}{K} \rceil$ and $\lceil \frac{i}{A(K_i,G)} \rceil \leq \lceil \frac{i}{\lceil \frac{i}{K} \rceil} \rceil \leq K$. Furthermore $A(K_K,G) \leq 1 \cdot g(K,|v_G|)$, so $A'(G) \geq \lceil \frac{K}{g(K,|V_G|)} \rceil$. This shows $R_{A'}(G) \leq g(K,|V_G|)$. □

Corollary 7.6.5.3. R_{min} (CLIQUE) $\leq R_{min}$ (MCE).

7.6.6. Relation of MCE with BALANCED COMPLETE BIPARTITE SUBGRAPH. Similar results can be obtained, relating approximation algorithms for MCE to approximation algorithms for the BALANCED COMPLETE BIPARTITE SUBGRAPH problem (BCBS). Again, a (bipartite) graph $G=(V_G,E_G)$ is said to contain a BCBS with 2*K nodes, if there are disjunct $V_1,V_2 \subseteq V_G$, $|V_1|=|V_2|=K$ with for all $v_1 \in V_1$, $v_2 \in V_2$ $(v_1,v_2) \in E_G$. Recall that the BCBS problem asks to find the largest BCBS in a given bipartite graph G and that BCBS is NP-complete [GJ79]. (Cf. section 7.5.).

Definition. The Balanced Complete Bipartite graph with 2*K nodes is the (undirected) graph $BCB_{2K} = (V_{2K},E_{2K})$, with $V_{2K} = \{v^i_j | i \in \{1,2\}, 1 \leq j \leq K\}$ and $E_{2K} = \{(v^1_{j_1}, v^2_{j_2}) \mid v^1_{j_1}, v^2_{j_2} \in V_{2K}\}$.

Lemma 7.6.6.1. Let $G = (V_G,E_G)$ be an undirected, bipartite graph and $K,c \in N^+$, $\lceil \frac{K}{c} \rceil \geq 2$. BCB_{2K} can be emulated on G with computation cost $\leq c$, if and only if G contains a BCBS with $2*\lceil \frac{K}{c} \rceil$ nodes.

Proof. Suppose f emulates BCB_{2K} on G with $cc(f) \leq c$. Take $V_1 = f(\{v^1_j \mid 1 \leq j \leq K\})$ and $V_2 = f(\{v^2_j \mid 1 < j \leq K\})$. Now $|V_1| \geq \lceil \frac{K}{c} \rceil \geq 2$, $|V_2| \geq \lceil \frac{K}{c} \rceil \geq 2$, and every node in V_1 is connected to every node in V_2. So V_1 and V_2 must be disjunct (G is bipartite) and G has a BCBS with 2*K nodes. (The remainder of the proof is more or less similar to lemma 7.6.5.1.) □

Corollary 7.6.6.2. Suppose there exists a polynomial time approximation algorithm A for MCE with $R_A(G,H) \leq g(|V_G|,|V_H|)$, for a certain function g. Then there exists a polynomial time approximation algorithm A' for BCBS with $R_{A'}(G) \leq g(2K \cdot |V_G|)$ for all connected, bipartite graphs G of which the largest BCBS consists of 2*K nodes.

Corollary 7.6.6.3. R_{min} (BCBS) ≤ R_{min} (MCE).

7.6.7. Discussion. The results in this section leave a number of questions open. For instance, we do not know whether $2 \leq R_{min}$ (MCE) $< \infty$ or R_{min} (MCE) $= \infty$, i.e. whether there is a polynomial time approximation algorithm for MINIMUM COST EMULATION that finds emulations whose computation costs differ at most a constant factor from the optimal computation cost. We gave some evidence that it will be hard to find such an algorithm: the existence of such an algorithm would imply existence of polynomial time approximation algorithms for BANDWIDTH, CLIQUE and BALANCED COMPLETE BIPARTITE SUBGRAPH that give approximations to within a constant factor from the optimal solution. Up to now, it is not know whether such algorithms exist.

7.7. The complexity of finding coverings. Recall the definition of covering from section 6.4.1. In this section we consider the following problem:

[GRAPH COVERING]
Instance: Connected, undirected graphs $G=(V_G,E_G)$, $H=(V_H,E_H)$
Question: Is there a covering of G on H?

The subproblem of GRAPH COVERING, in which the computation factor $c=|V_G|/|V_H|$ is fixed is called c-GRAPH COVERING. The following lemma shows that 1-GRAPH COVERING and GRAPH ISOMORPHISM are virtually the same problem.

Lemma 7.7.1. [Re32] Let $G=(V_G,E_G)$ and $H=(V_H,E_H)$ be connected, undirected graphs and $|V_G|=|V_H|$. A function $f\colon V_G \to V_H$ covers G on H iff f is a graph isomorphism of G on H.

It is still an open problem whether GRAPH ISOMORPHISM is solvable in polynomial time or not, and whether it is NP-complete or not [GJ79]. Theorem 7.7.2. shows that c-GRAPH COVERING is at least as hard as GRAPH ISOMORPHISM, indicating that it will be very hard, if not impossible to find a polynomial time algorithm for it: any polynomial time algorithm for c-GRAPH COVERING would enable us to solve GRAPH ISOMORPHISM in polynomial time.

Theorem 7.7.2. If there exists an algorithm that solves c-GRAPH COVERING in time $\le f(|V_H|)$, then there exists an algorithm that solves GRAPH ISOMORPHISM in time $\le f(|V_H|+4)+ O(\max\{|V_H|, |E_G|, |E_H|\})$, for every $c\ge 1$.

Proof.

For $c=1$ this follows from lemma 7.7.1. Let $c\ge 2$ be given, and let there exists an algorithm for c-GRAPH COVERING that uses time $\le f(|V_H|)$, that is: it decides whether $G=(V_G,E_G)$ covers $H=(V_H,E_H)$ in time at most $f(|V_H|)$.

Now let an instance of GRAPH ISOMORPHISM be given, i.e. we have two connected graphs $G=(V_G,E_G)$ and $H=(V_H,E_H)$, with $|V_G|=|V_H|$ and $|E_G|=|E_H|$ and ask the question whether G and H are graph isomorphic.

We will now define graphs G′, H′ with $|V_{G'}|=|V_H|+4$, $|V_{G'}|=c.|V_{H'}|$ and G′ covers H′ if and only if G and H are graph isomorphic.

To H we add 4 extra nodes v_1', v_2', v_3', v_4', with extra edges between v_1' and every node in V_H, and (v_1', v_2'), (v_2', v_3'), (v_3', v_4'), and (v_4', v_2'). (See fig. 7.7.1.)

G′ is formed by taking c copies of G, adding 4c extra nodes v_j^i $(1\le i\le c,\ 1\le j\le 4)$, and adding the following edges:

- v_1^i is connected to every node in the i^{th} copy of G $(1\le i\le c)$
- v_1^i is connected to v_2^i $(1\le i\le c)$
- v_2^i is connected to v_3^i and to v_4^i $(1\le i\le c)$
- v_3^i is connected to $v_4^{(i+1)\bmod c}$ $(1\le i\le c)$

An example of the construction is shown in fig 7.7.2., with c=4.

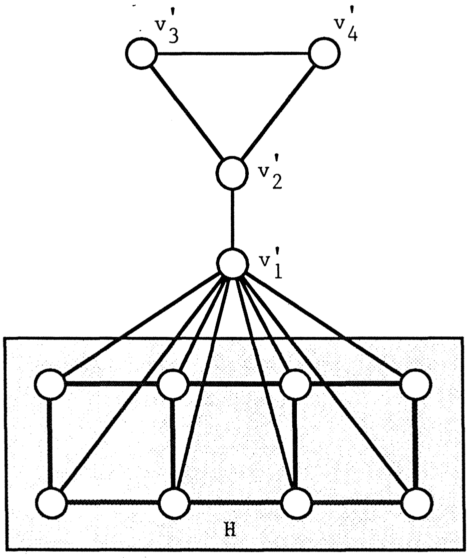


fig. 7.7.1.

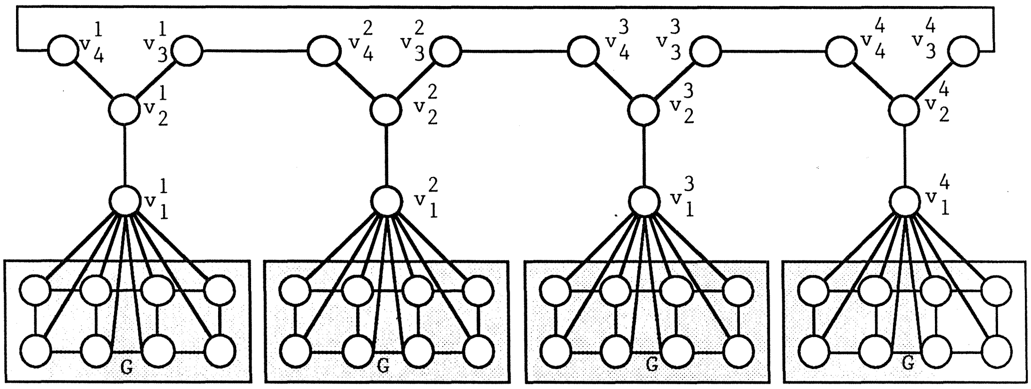


fig 7.7.2. G′ with c=4

Claim 7.7.2.1. G′ covers H′ if and only if G and H are graph isomorphic.

Proof.

Let f cover G′ on H′. Note that the only node with degree $|V_H|+1$ in H′ is v_1'. Hence $f(v_1^i) = v_1'$ for every i, $1 \le i \le c$. This shows that the copy of G, connected to v_1^1, must be mapped upon the copy of H, connected to v_1' and f restricted to this copy of G gives a graph isomorphism of G on H.

It is easy to see that if G and H are graph isomorphic, then G′ covers H′. □

G′ and H′ can be constructed in time $O(\max\{|V_H|, |E_G|, |E_H|\})$, hence there exists an algorithm that solves GRAPH ISOMORPHISM in time ≤ $f(|V_H|+4) + O(\max\{|V_H|, |E_G|, |E_H|\})$. □

It is an interesting open question whether c-GRAPH COVERING is polynomially equivalent to GRAPH ISOMORPHISM for $c \ge 2$, and whether it is NP-complete.

7.8. A note on the complexity of finding uniform emulations of networks with buses. The problem to decide whether there exist computationally uniform emulations from a given hypergraph G on a given hypergraph H is, in general, NP-complete. For instance, this problem contains the UNIFORM EMULATION problem for graphs as a special case (cf. sections 7.2. -7.5.). One also has that the problem whether a given hypergraph can be emulated with computation cost 1 on a 2 dimensional spanning bus hypercube, with given width N in one, and M in the other dimension is NP-complete: it contains as a special case the EDGE EMBEDDING ON A GRID problem. (This problem is: given a graph G=(V,E), positive integers M,N, is there an injective function $f: V \to \{1,2,\dots,M\} \times \{1,2,\dots,N\}$, such that if $(u,v) \in E$ then $f(u)=(x_1,x_2)$, $f(v)=(y_1,y_2)$ implies $x_1=x_2$ or $y_1=y_2$, i.e. f(u) and f(v) are on the same 'line' in the grid?) EDGE EMBEDDING ON A GRID is NP-complete (see e.g. [GJ79,p.219]).

Appendix A

Proof of estimates for $K_n(2)$ and $K_n(3)$ (lemma 2.2.3.2.)

We give a proof of the estimates for $K_n(2)$ and $K_n(3)$ stated in lemma 2.2.3.2. through the following auxiliary results.

Lemma 2.2.3.2.1. $K_n(2) = (\frac{1}{2}-\frac{1}{2}\ln 2)H_n + O(1)$.

Proof.

We use the following estimate: if $f(x)$ is non-negative and decreasing on $[a,b]$, then $\sum_{t=a}^{b} f(t) = \int_a^b f(x)dx+\varepsilon f(a)$ for some $0\leq\varepsilon\leq 1$. Taking $f(x) = \frac{t_1-\frac{1}{2}x}{(x-1)x}$ we obtain

$$\sum_{x=t_1+1}^{2t_1-1} \frac{t_1-\frac{1}{2}x}{(x-1)x} = \int_{t_1+1}^{2t_1-1} \frac{t_1-\frac{1}{2}x}{(x-1)x}\,dx + O(\frac{1}{t_1}) =$$

$$= \left[(t_1-\tfrac{1}{2})\ln(x-1)-t_1\ln x\right]\Big|_{x=t_1+1}^{x=2t_1-1} +O(\frac{1}{t_1}) =$$

$$= \frac{1}{2}-\frac{1}{2}\ln 2+O(\frac{1}{t_1}).$$

Now observe from the definition of $K_n(2)$ that

$$K_n(2)= \sum_{\substack{1<t_1<t_2\leq n\\ t_2<2t_1}} \frac{t_1-\frac{1}{2}t_2}{(t_1-1)(t_2-1)t_2} = \sum_{t_1=2}^{n-1}\ \sum_{t_2=t_1+1}^{\min(n,2t_1-1)} \frac{t_1-\frac{1}{2}t_2}{(t_1-1)(t_2-1)t_2}$$

and (thus)

$$\sum_{t_1=2}^{\frac{1}{2}n}\ \sum_{t_2=t_1+1}^{2t_1-1} \frac{t_1-\frac{1}{2}t_2}{(t_1-1)(t_2-1)t_2} \leq K_n(2)\leq \sum_{t_1=2}^{n-1}\ \sum_{t_2=t_1+1}^{2t_1-1} \frac{t_1-\frac{1}{2}t_2}{(t_1-1)(t_2-1)t_2}.$$

By substituting the previous estimate we obtain

$$(\tfrac{1}{2}-\tfrac{1}{2}\ln 2)\sum_{t_1=2}^{\frac{1}{2}n}\frac{1}{t_1-1}+O\left(\sum_{t_1=2}^{\frac{1}{2}n}\frac{1}{t_1^2}\right)\le K_n(2)\le(\tfrac{1}{2}-\tfrac{1}{2}\ln 2)\sum_{t_1=2}^{n-1}\frac{1}{t_1-1}+O\left(\sum_{t_1=2}^{n-1}\frac{1}{t_1^2}\right)$$

, hence $K_n(2) = (\frac{1}{2}-\frac{1}{2}\ln 2)\ H_n + O(1)$. □

To derive an estimate for $K_n(3)$ we follow a similar approach, although the analysis becomes slightly more involved.

Lemma 2.2.3.2.2. For $t_1 \le u < 2t_1-1$, $\displaystyle\sum_{t=u+1}^{2t_1-1}\frac{t_1-\frac{1}{2}x}{(t-1)t} = \frac{t_1}{u} - \frac{1}{2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\frac{t_1}{u}$ $+ O(\frac{1}{t_1})$.

Proof.

$$\sum_{t=u+1}^{2t_1-1}\frac{t_1-\frac{1}{2}t}{(t-1)t} = \int_{u+1}^{2t_1-1}\frac{t_1-\frac{1}{2}x}{(x-1)x}\,dx + O(\frac{1}{t_1}) =$$

$$= \left[(t_1-\tfrac{1}{2})\ln(x-1)-t_1\ln x\right]\Big|_{x=u+1}^{x=2t_1-1} + O(\frac{1}{t_1}) =$$

$$= (t_1-\tfrac{1}{2})\ln(2t_1-2)-t_1\ln(2t_1-1)+t_1\ln(u+1)-(t_1-\tfrac{1}{2})\ln u+O(\frac{1}{t_1}) =$$

$$= \frac{t_1}{u} - \frac{1}{2} - \frac{1}{2}\ln 2 - \frac{1}{2}\ln\frac{t_1}{u} + O(\frac{1}{t_1}),$$

where we use that $\log(1+z) = z + O(z^2)$ for $z \le \frac{1}{2}$. □

It follows that

$$\sum_{t_1=3}^{n}\ \sum_{t_2=t_1+1}^{2t_1-2}\ \sum_{t_3=t_2+1}^{2t_1-1}\frac{t_1-\frac{1}{2}t_3}{(t_1-1)(t_2-1)(t_3-1)t_3} =$$

$$\sum_{t_1=3}^{n}\ \sum_{t_2=t_1+1}^{2t_1-2}\frac{\frac{t_1}{t_2}-\frac{1}{2}-\frac{1}{2}\ln 2-\frac{1}{2}\ln\frac{t_1}{t_2}}{(t_1-1)(t_2-1)} + \sum_{t_1=2}^{n}\ \sum_{t_2=t_1+1}^{2t_1-2}\frac{O\left(\frac{1}{t_1}\right)}{(t_1-1)(t_2-1)}.$$

Note that $\displaystyle\sum_{t_2=t_1+1}^{2t_1-2}\frac{1}{t_2-1}$ consists of t_1-2 terms of size $\le \frac{1}{t_1}$ and thus is

O(1). Consequently the second summand in the expression above is $O\left(\sum_{t_1\geq 2}\frac{1}{t_1^2}\right)=O(1)$. Rewrite the first summand as B-C, with

$$B=\sum_{t_1=3}^{n}\sum_{t_2=t_1+1}^{2t_1-2}\frac{t_1-\frac{1}{2}t_2}{(t_1-1)(t_2-1)t_2},$$

$$C=\sum_{t_1=3}^{n}\sum_{t_2=t_1+1}^{2t_1-2}\frac{\frac{1}{2}\ln2+\frac{1}{2}\ln\frac{t_1}{t_2}}{(t_1-1)(t_2-1)}.$$

Lemma 2.2.3.2.3. For $a\geq\frac{1}{2}\ln2+\frac{1}{2}\ln t_1$, $\sum_{t=t_1+1}^{2t_1-2}\frac{a-\frac{1}{2}\ln t}{(t-1)}=a\ln2-\frac{1}{4}\ln^2 2-\frac{1}{2}\ln2\cdot\ln t_1+O(\frac{a}{t_1})$.

Proof.

$$\sum_{t=t_1+1}^{2t_1-2}\frac{a-\frac{1}{2}\ln t}{(t-1)}=\sum_{t=t_1+1}^{2t_1-2}\frac{a-\frac{1}{2}\ln t}{t}+\sum_{t=t_1+1}^{2t_1-2}\frac{a-\frac{1}{2}\ln t}{t(t-1)}=$$

$$=\int_{t_1+1}^{2t_1-2}\frac{a-\frac{1}{2}\ln x}{x}\,dx+O(\frac{a}{t_1})=$$

$$=(a\ln x-\frac{1}{4}\ln^2 x)\Big|_{x=t_1+1}^{x=2t_1-2}+O(\frac{a}{t_1})=$$

$$=a\ln2-\frac{1}{4}\ln^2 2-\frac{1}{2}\ln2\cdot\ln t_1+O(\frac{a}{t_1}).\quad\square$$

Lemma 2.2.3.2.4. $K_n(3)=(\frac{1}{2}-\frac{1}{2}\ln2-\frac{1}{4}\ln^2 2)H_n+O(1)$.

Proof.

By the analysis in the proof of lemma 2.2.3.2.1. it easily follows that we have

$$B = \sum_{t_1=3}^{n} \frac{\frac{1}{2}-\frac{1}{2}\ln 2+O(\frac{1}{t_1})}{(t_1-1)} = (\tfrac{1}{2}-\tfrac{1}{2}\ln 2)H_n + O(1).$$

Applying lemma 2.2.3.2.3. with $a=\frac{1}{2}\ln 2 + \frac{1}{2}\ln t_1$ we obtain

$$C = \sum_{t_1=3}^{n} \frac{\frac{1}{4}\ln^2 2 + O\left[\frac{\ln t_1}{t_1}\right]}{(t_1-1)} = \frac{1}{4}\ln^2 2\ H_n + O(1),$$

and thus $\sum\limits_{t_1=3}^{n} \sum\limits_{t_2=t_1+1}^{2t_1-2} \sum\limits_{t_3=t_2+1}^{2t_1-1} \frac{t_1-\frac{1}{2}t_3}{(t_1-1)(t_2-1)(t_3-1)t_3} = B-C+O(1) =$

$= (\frac{1}{2} - \frac{1}{2}\ln 2 - \frac{1}{4}\ln^2 2)H_n + O(1)$. As in the proof of lemma 2.2.3.2.1 one can now estimate $K_n(3)$ by this expression from above, and also from below using $\frac{1}{2}n$ instead of n. Thus $K_n(3) = (\frac{1}{2} - \frac{1}{2}\ln 2 - \frac{1}{4}\ln^2 2)$ $H_n+O(1)$. □

Appendix B. The proof of the Characterization Theorem for the uniform emulations of S_n on S_{n-1} (theorem 5.3.2.5.).

We use the notations and terminology of Section 5.3. Our aim is to prove the following result.

Theorem 5.3.2.5. (Characterization Theorem). Every uniform emulation of S_n on S_{n-1} is step-simulating, and thus equal to one of the mappings listed in table A.

The proof is based on the lemma below and a subsequent analysis of cases. Assume $n>2$.

For the proof of theorem 5.3.2.5., assume that there exists a uniform emulation f of S_n on S_{n-1} that is not step-simulating. It follows that there must be an $x \in (\frac{o}{1})^{n-1}$, $y \in (\frac{o}{1})^{n-3}$ and $\alpha,\beta,\gamma,\delta \in (\frac{o}{1})$ such that $f(\alpha x) = \beta y\delta$ and $f(x\gamma) = \beta y\delta$, with $\beta y \neq y\delta$. (Cf. lemma 5.3.1.3. and lemma 5.3.2.2.) We will fix the notation throughout the remainder of this section.

Claim 5.3.2.5.1. Under the assumption stated, one of the following situations must hold

(i) $x = o^{n-1}$ and $(\alpha=o \vee \gamma=o)$

(ii) $x = 1^{n-1}$ and $(\alpha=1 \vee \gamma=1)$

(iii) $\beta y\delta = (o1)*[o]$

(iv) $\beta y\delta = (1o)*[1]$

Proof.

In addition to $f(\alpha x) = f(x\gamma) = \beta y\delta$ we must have : $(f(\bar{\alpha}x) = \beta y\delta \vee f(\bar{\alpha}x) = \frac{o}{1}\beta y)$ and $(f(x\bar{\gamma}) = \beta y\delta \vee f(x\bar{\gamma}) = y\delta\frac{o}{1})$, from the emulation property. Because f is uniform, only two nodes can be mapped to $\beta\gamma\delta$. The following situation can be distinguished :

(a) $f(\bar{\alpha}x) = f(\alpha x) = \beta y\delta$. Because $f(x\gamma) = \beta y\delta$ also, we have $x\gamma = \bar{\alpha}x$ ($\Rightarrow$ $x=o^{n-1}$ and $\alpha=1$ and $\gamma=o$, or $x=1^{n-1}$ and $\alpha=o$ and $\gamma=1$) or $x\gamma = \alpha x$ ($\Rightarrow$ $x=o^{n-1}$ and $\alpha=o$ and $\gamma=o$, or $x=1^{n-1}$ and $\alpha=1$ and $\gamma=1$).

(b) $f(x\bar{\gamma}) = f(x\gamma) = \beta y\delta$. Now also $f(\alpha x) = \beta y\delta$, and the same cases as under (a) result.

(c) $f(\bar{\alpha}x) = \frac{o}{1}\beta y$ and $f(x\bar{\gamma}) = y\delta\frac{o}{1}$. Clearly $f(x\bar{\gamma}) = \frac{o}{1}\beta y$ or $f(x\bar{\gamma}) = \beta y\bar{\delta}$, hence $\frac{o}{1}\beta y = y\delta\frac{o}{1}$ or $\beta y\overline{\delta} = y\delta\frac{o}{1}$. Because $\beta y \neq y\delta$ only the former case can arise : $\frac{o}{1}\beta y = y\delta\frac{o}{1}$. It follows that $\beta y\delta$ = (o1)*[o] or $\beta y\delta$ = (1o)*[1]. (The "solutions" $\beta y\delta = o^{n-1}$ and $\beta y\delta = 1^{n-1}$ are not valid, because it would yield $\beta y = y\delta$.) □

We now obtain the basic step for the further case analysis.

Lemma 5.3.2.5.2. Under the assumption stated, one of the following six cases must hold :

(I) f((o1)*[o]) = o^{n-1}, f((1o)*[1]) = o^{n-1}
(II) f((o1)*[o]) = 1^{n-1}, f((1o)*[1]) = 1^{n-1}
(III) f((o1)*[o]) = (o1)* [o], f((1o)*[1]) = (o1)*[o]
(IV) f((o1)*[o]) = (o1)* [o], f((1o)*[1]) = (1o)*[1]
(V) f((o1)*[o]) = (1o)* [1], f((1o)*[1]) = (o1)*[o]
(VI) f((o1)*[o]) = (1o)* [1], f((1o)*[1]) = (1o)*[1].

Proof.

Let f((o1)*[o]) = $u_1..u_{n-1}$ and f((1o)*[1]) = $v_1..v_{n-1}$. Because (o1)*[o] and (1o)*[1] are adjacent in S_n and f is an emulation, the following situations can arise :

(a) $u_1..u_{n-1} = v_1..v_{n-1}$. Write $u_1..u_{n-1} = \beta y\delta$. (Note that we cannot assume that $\beta y \neq y\delta$.) By the analysis under claim 5.3.2.5.1. it follows that $\frac{o}{1}\beta y = y\delta\frac{o}{1}$ (hence $\beta y\delta = o^{n-1}$, 1^{n-1}, (o1)*[o], or (1o)*[1]) or $\beta y\overline{\delta} = y\delta\frac{o}{1}$ (hence $\beta y\delta = o^{n-1}$ or 1^{n-1}). This proves cases I, II, III, and VI.

(b) $u_1..u_{n-1} \neq v_1..v_{n-1}$, but $u_1..u_{n-1} = v_2..v_{n-1}\frac{o}{1} = \frac{o}{1}v_1..v_{n-2}$. It follows that $v_1..v_{n-1} = o^{n-1}$, 1^{n-1}, (o1)*[o], (1o)*[1] but only for the latter two cases can $u_1..u_{n-1}$ be chosen to satisfy the constraint (namely $u_1..u_{n-1}$ =(1o)*[1], (o1)*[o] respectively). This proves cases IV and V. □

We proceed by analyzing the cases of lemma 5.3.2.5.2. and showing that in each case a contradiction must arise. (Recall the assumption that f is uniform and not step-simulating.)

Case I. f((o1)*[o]) = f((1o)*[1]) = o^{n-1}.

We show that this forces f to be equal to f_3, one of the six step-simulations listed in table A.

Claim 5.3.2.5.3. For $1 \le i \le n-1$ and $b \in (\frac{o}{1})^n$, $f_i(b_1..b_n) = (b_i \equiv b_{i+1})$.

Proof.

Define $B^n_i = \{b_1..b_n \mid \forall_j : 1 \le j \le i-1 \Rightarrow b_j \ne b_{j+1}\} \subseteq (\frac{o}{1})^n$ and $C^{n-1}_{i-1} = \{c_1..c_{n-1} \mid \forall_j : 1 \le j \le i-1 \Rightarrow c_j = o\} \subseteq (\frac{o}{1})^{n-1}$. Note that $B^n_n = \{(o1)*[o], (10)*[1]\}$ and $C^{n-1}_{n-1} = \{o^{n-1}\}$, and hence that $f(B^n_n) = C^{n-1}_{n-1}$ and $f^{-1}(C^{n-1}_{n-1}) = B^n_n$ (by uniformity). We claim that for all $1 \le i \le n$, $f(B^n_i) = C^{n-1}_{i-1}$ and $f^{-1}(C^{n-1}_{i-1}) = B^n_i$. For a proof, use downward induction starting with $i=n$, for which the claim clearly holds. Suppose it holds for some $i \ge 1$. Consider any $b_1..b_n \in B^n_{i-1}$. It follows that $\bar{b}_1 b_1..b_{n-1} \in B^n_i$ and thus that $f(\bar{b}_1 b_1..b_{n-1}) \in C^{n-1}_{i-1}$. Since f is an emulation, we must have $f(b_1..b_n) \in C^{n-1}_{i-1}$ or $f(b_1..b_n) \in C^{n-1}_{i-2}$. In either case $f(b_1..b_n) \in C^{n-1}_{i-2}$, and we have $f(B^n_{i-1}) \subseteq C^{n-1}_{i-2}$. Because $|B^n_{i-1}| = 2|C^{n-1}_{i-2}|$ and f is uniform, we have in fact $f(B^{n-1}_{i-1}) = C^{n-1}_{i-2}$ and ipso facto $f^{-1}(C^{n-1}_{i-2}) = B^n_{i-1}$. This completes the inductive argument.

We immediately conclude (take $i=2$) that for all $x \in (\frac{o}{1})^{n-2}$, $f_1(o1x) = o$ and $f_1(1ox) = o$. Because of uniformity this forces $f_1(oox) = f_1(11x) = 1$ for all $x \in (\frac{o}{1})^{n-2}$. Define $\tilde{B}^n_i = \{b_1..b_n \mid b_n..b_1 \in B^n_i\}$ and $\tilde{C}^{n-1}_{i-1} = \{c_1..c_{n-1} \mid c_{n-1}..c_1 \in C^{n-1}_{i-1}\}$. As before one shows that for all $1 \le i \le n$, $f(\tilde{B}^n_i) = \tilde{C}^{n-1}_{i-1}$ and $f^{-1}(\tilde{C}^{n-1}_{i-1}) = \tilde{B}^n_i$. We now argue by downward induction on i that for all $x \in \tilde{B}^n_i$, $f(x) = f_3(x)$ (with f_3 as in table A). For $i=n$ we have $\tilde{B}^n_n = \{(o1)*[o], (1o)*[1]\}$ and $f((o1)*[o]) = f((1o*)[1]) = o^{n-1}$, which indeed coincides with f_3. Suppose it holds for some $i \ge 1$. Consider any $x_1..x_{n-i+1}(o1)*[o] \in \tilde{B}^n_{i-1}$. If $x_{n-i+1} = 1$ then f and f_3 coincide on the argument by induction. Let $x_{n-i+1} = o$. Observe that $f(x_2..x_{n-i+1}(o1)*[o]) \in f(\tilde{B}^n_i) = \tilde{C}^{n-1}_{i+1}$ and that $f(x_1..x_{n-i+1}(o1)*[o]) \in f(\tilde{B}^n_{i-1} \backslash \tilde{B}^n_i) = \tilde{C}^{n-1}_{i-2} \backslash \tilde{C}^{n-1}_{i-1}$, where the latter holds because $x_{n-i+1} = o$ and uniformity of f. It follows that $f(x_1..x_{n-i+1}(o1)*[o]) \ne f(x_2..x_{n-i+1}(o1)*[o])$ and thus, because f is an emulation, necessarily $f(x_1..x_{n-i+1}(o1)*[o]) = \frac{o}{1}.f(x_2..x_{n-i+1}(o1)*[o])|_{n-2}$. Using the inductive assertion it follows that $f_i(x_1..x_{n-i+1}(o1)*[o]) = (f_3)_i(x_1..x_{n-i+1}(o1)*[o])$ for all $2 \le i \le n-1$. At the beginning of this paragraph we showed that this must hold also for $i=1$. Thus f and f_3 coincide on $\tilde{B}^n_{i-1}$, which completes the inductive argument. Because $\tilde{B}^n_1 =$

V_n this shows that f and f_3 coincide for all arguments, which proves the claim. □

Because f was assumed nòt to be step-simulating, claim 5.3.2.5.3. clearly proves that case I is contradictory.

<u>Case II</u>. $f((o1)*[o]) = f((1o)*[1]) = 1^{n-1}$.
The proof of claim 5.3.2.5.3. can be completely dualized to show that in this case f must be equal to $\overline{f}_3$, another one of the six step-simulations listed in table A. Because f was assumed nòt to be step-simulating, this case is also contradictory.

<u>Case III</u>. $f((o1)*[o]) = f((1o)*[1]) = (o1)*[o]$.
We show that for n>2 no emulation f of S_n on S_{n-1} with this property exists. Suppose on the contrary that an f does exist. We derive a contradiction as follows.

First let n be odd, which implies that the assumption turns into $f((o1)*o) = f((1o)*1) = (o1)*$. Since f is uniform no other nodes can be mapped to (o1)*, and we necessarily obtain: $f(oo(1o)*1) = \frac{o}{1}(o1)*o$, $f(1(o1)*oo) = 1(o1)*\frac{o}{1}$, $f(11(o1)*o) \in \{1(o1)*\frac{o}{1}, 11(o1)*\}$, $f(o(1o)*11) \in \{\frac{o}{1}(o1)*o, (o1)*oo\}$. Observing that necessarily $(f(11(o1)*o), f((1o)*1)) \in E_{n-1}$ and $(f((1o)*1), f(o(1o)*11)) \in E_{n-1}$ it follows that $f(11(o1)*o) = 1(o1)*\frac{o}{1}$ and $f(o(1o)*11) = \frac{o}{1}(o1)*o$, and the emulation property now forces that $f(1(o1)*oo) = f(11(o1)*o) = 1(o1)*\frac{o}{1}$ and $f(oo(1o)*1) = f(o(1o)*11) = \frac{o}{1}(o1)*o$. From the assumption one easily derives $f(11(o1)*o) = \frac{o}{1}(o1)*o$ and $f(o(1o)*11) = 1(o1)*\frac{o}{1}$, thus forcing all four nodes to be mapped to 1(o1)*o. This contradicts uniformity.

For n even we have $f((o1)*) = f((1o)*) = (o1)*o$. By uniformity again no other nodes are mapped to (o1)*o, and we necessarily obtain: $f(oo(1o)*) = \frac{o}{1}(o1)*$, $f((o1)*oo) = (1o)*\frac{o}{1}$, $f((1o)*11) = (1o)*\frac{o}{1}$, $f(11(o1)*) = \frac{o}{1}(o1)*$. The emulation property forces f(oo(1o)*) and f((o1)*oo) to be adjacent in S_{n-1} (impossible) or equal, hence $f(oo(1o)*) = f((o1)*oo) = 1(o1)*$. By the same argument $f((1o)*11) = f(11(o1)*) = 1(o1)*$. Thus four nodes are mapped to 1(o1)*, contradicting uniformity.

Case IV. $f((o1)*[o]) = (o1)*[o]$, $f((1o)*[1]) = (1o)*[1]$.
A more tedious argument is required to show that in this case again every uniform emulation f that satisfies the constraint must be step-simulating, contrary to our basic assumption.

First let n=3, which turns the constraint into $f(o1o) = o1$ and $f(1o1) = 1o$. We show that f must be equal to the step-simulations f_1 or $\overline{f}_2$ from table A. By emulation $f(oo1) \in \{o1,oo,1o\}$, $f(1oo) \in \{o1,1o,11\}$, $f(o11) \in \{1o,oo,o1\}$, $f(11o) \in \{1o,o1,11\}$. Uniformity is heavily used in the following further analysis:

(a) Suppose $f(oo1) = f(o1o) = o1$. Then $f(1oo) = \frac{o}{1}o = 1\frac{o}{1}$, hence $f(1oo) = f(1o1) = 1o$. It follows that $f(o11) = 1\frac{o}{1} = o\frac{o}{1}$, contradiction.

(b) Suppose $f(oo1) = oo$. It follows that $f(1oo) = 1o$ $(=f(1o1))$ and $f(ooo) \in \{oo,1o\}$, hence $f(ooo) = f(oo1) = oo$ and thus $f(o11) = o1$ $(=f(o1o))$ and $f(11o) = 11$. Necessarily $f(111) = 11$, and f is proved to coincide with f_1 from table A.

(c) Suppose $f(oo1) = f(1o1) = 1o$. It follows that $f(1oo) \in \{o1,11\}$. If $f(1oo) = f(o1o) = o1$, then $f(11o) = oo$ and this is impossible. Thus $f(1oo) = 11$ and necessarily $f(11o) = o1$ $(=f(o1o))$ and $f(o11) = oo$. It follows by emulation that $f(ooo) = f(1oo) = 11$, and $f(111) = oo$. This proves f equal to $\overline{f}_2$ from table A.

Now let $n\geq4$. We shall first derive a number of auxiliary facts that are needed later.

Claim 5.3.2.5.4. For $n\geq4$, $f(o^n) \in \{o^{n-1},1^{n-1}\}$ and $f(1^n) \in \{o^{n-1},1^{n-1}\}$.
Proof.

We only consider $f(o^n)$, as the argument for $f(1^n)$ is similar. Let $f(o^n) = u_1..u_{n-1}$. Then $f(1o^{n-1}) \in \{u_1..u_{n-1},\frac{o}{1}u_1..u_{n-2}\}$ and $f(o^{n-1}1) \in \{u_1..u_{n-1},u_2..u_{n-1}\frac{o}{1}\}$. The following cases can arise :

(a) $f(1o^{n-1}) = f(o^n) = u_1..u_{n-1}$. Because of uniformity we must have that $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1} \neq u_1..u_{n-1}$, and also $f(o1o^{n-2}) = \frac{o}{1}u_1..u_{n-2}$ and $f(1o^{n-2}1) \in \{\frac{o}{1}u_1..u_{n-2},\ u_1..u_{n-2}\frac{o}{1}\}$. If $f(1o^{n-2}1) = f(o1o^{n-2}) = \frac{o}{1}u_1..u_{n-2}$ then by uniformity $f(o^{n-2}1o) = u_1..u_{n-2}\frac{o}{1}$ and $f(o^{n-2}11) = u_1..u_{n-2}\frac{o}{1}$, hence $f(o^{n-2}1o) = f(o^{n-2}11) = u_1..u_{n-2}\bar{u}_{n-1}$. Thus $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1} = \frac{o}{1}u_1..u_{n-2}$ and necessarily $u = o^{n-1}$ or $u = 1^{n-1}$. In either case uniformity is contradicted. Thus $f(1o^{n-2}1) = u_1..u_{n-2}\frac{o}{1}$, which implies in fact that $f(1o^{n-2}1) = u_1..u_{n-2}\bar{u}_{n-1}$ and hence $f(o^{n-2}1o) \in \{u_1..u_{n-2}\bar{u}_{n-1}, u_2..u_{n-2}\bar{u}_{n-1}\frac{o}{1}\}$. If $f(o^{n-2}1o) = f(1o^{n-2}1) = u_1..u_{n-2}\bar{u}_{n-1}$ then $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1} = \frac{o}{1}u_1..u_{n-2}$ and necessarily $u = o^{n-1}$ or $u = 1^{n-1}$. In either case uniformity is contradicted again. Thus $f(1o^{n-2}1) = u_1..u_{n-2}\bar{u}_{n-1}$ and $f(o^{n-2}1o) = u_2..u_{n-2}\bar{u}_{n-1}\frac{o}{1}$, and thus $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1} = \frac{o}{1}u_2..u_{n-2}\bar{u}_{n-1}$. It follows that $u_2..u_{n-1} = \alpha^{n-2}$ (with $\alpha = o$ or $\alpha = 1$) and $f(o^{n-1}1) = \alpha^{n-2}\bar{\alpha}$. If $u_1 = \alpha$ then we are finished. Thus assume that $u_1 = \bar{\alpha}$, hence $f(o^n) = f(1o^{n-1}) = \bar{\alpha}\alpha^{n-2}$. Consider $b_1..b_n \in f^{-1}(\alpha^{n-1})$, thus $f(b_1..b_n) = \alpha^{n-1}$. Because of uniformity it follows that $f(ob_1..b_{n-1}) = f(1b_1..b_{n-1}) = \alpha^{n-1}$ (using that $b_1..b_{n-1} \neq o^{n-1}$), and likewise $f(o1b_1..b_{n-2}) = f(11b_1..b_{n-2}) = \alpha^{n-1}$ and, provided $b_1..b_{n-2} \neq o^{n-2}$, also $f(oob_1..b_{n-2}) = \alpha^{n-1}$. This shows that at least 3 nodes are mapped to α^{n-1}, contradicting uniformity.

(b) $f(o^{n-1}1) = f(o^n) = u_1..u_{n-1}$. The argument is analogous to case (a) by 'reversing' the orientation of the strings.

(c) $f(1o^{n-1}) = \frac{o}{1}u_1..u_{n-2}$ and $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1}$. If $f(1o^{n-1}) = f(o^{n-1}1)$ then necessarily $u_1..u_{n-1} = (\alpha\beta)*[\alpha]$. Because of uniformity $\alpha=\beta$ (otherwise one of $(o1)*[o]$ and $(o1)*[1]$ would be mapped to $(\alpha\beta)*[\alpha]$ too), and thus $u_1..u_{n-1} = o^{n-1}$ or $u_1..u_{n-1} = 1^{n-1}$. It follows that $f(1o^{n-1}) = f(o^{n-1}1) = f(o^n)$, contradicting uniformity. Thus $f(1o^{n-1}) \neq f(o^{n-1}1)$ and by emulation necessarily $f(o^{n-1}1) = u_2..u_{n-1}\frac{o}{1} = u_1..u_{n-2}\frac{o}{1}$, hence $u_1..u_{n-1} = o^{n-1}$ or $u_1..u_{n-1} = 1^{n-1}$. □

(The condition $n\geq4$ was used in case (a), to make sure that $o^{n-2}1o \neq o1o^{n-2}$ and (hence) $1o^{n-2}1 \neq (1o)*[1]$ and $o^{n-2}\,1o \neq (o1)*[o]$.) Next observe from $f((o1)*[o]) = (o1)*[o]$ that $f(oo(1o)*[1]) \in \{(o1)*[o], (1o)*[1], oo(1o)*[1]\}$ and from $f((1o)*[1]) = (1o)*[1]$ that $f(11(o1)*[o]) \in \{o1)*[o], (1o)*[1], 11(o1)*[o]\}$.
We tackle a particular combination first, because it will be central in the remainder of the proof.

<u>Claim 5.3.2.5.5</u>. For $n\geq4$,

(i) if $f(oo(1o)*[1]) = oo(1o)*[1]$, then for all $b_1..b_{n-3} \in (\frac{o}{1})^{n-3}$ there exist $c_1..c_{n-3}$, $c'_1..c'_{n-3}$, $\in (\frac{o}{1})^{n-3}$ such that $f(b_1..b_{n-3}ooo) = c_1..c_{n-3}oo$ and $f(b_1..b_{n-3}oo1) = c'_1..c'_{n-3}oo$.

(ii) if $f(11(o1)*[o]) = 11(o1)*[o]$, then for all $b_1..b_{n-3} \in (\frac{o}{1})^{n-3}$ there exist $c_1..c_{n-3}$, $c'_1..c'_{n-3}$, $\in (\frac{o}{1})^{n-3}$ such that $f(b_1..b_{n-3}11o) = c_1..c_{n-3}11$ and $f(b_1..b_{n-3}111) = c_1..c_{n-3}11$.

Proof.

We only prove (i), as (ii) is similar. First we induct on i to show that for all $b_1..b_i \in (\frac{o}{1})^i$ there exist a $c_1..c_i \in (\frac{o}{1})^i$ with $f(b_1..b_i oo1(o1)*[o]) = c_1..c_i oo(1o)*[1]$. Since $f(oo1(o1)*[o]) = oo(1o)*[1]$ by assumption, we have for $i=1$: $f(b_1 oo1(o1)*[o]) \in \{oo1(o1)*[o], \frac{o}{1}oo(1o)*[1]\}$. If $f(b_1 oo1(o1)*[o]) = f(oo1(o1)*[o]) = oo(1o)*[1]$ then one easily verifies that claim 5.3.2.5.1. is contradicted. (Use $\alpha=b_1$, $x=oo1(o1)*[o]$, $\gamma=o$ or 1.) Thus $f(b_1 oo1(o1)*[o]) = c_1 oo(1o)*[1]$, for some $c_1 \in \frac{o}{1}$. Suppose it holds for some i, $1 \le i<n-3$. Consider $f(b_1..b_{i+1}oo1(o1)*[o])$. By induction there exists a $c_2..c_{i+1} \in (\frac{o}{1})^i$ such that $f(b_2..b_{i+1}oo1(o1)*[o]) = c_2..c_{i+1}oo(1o)*[1]$, and thus $f(b_1b_2..b_{i+1}oo1(o1)*[o]) \in \{c_2..c_{i+1}oo(1o)*[1], \frac{o}{1}c_2..c_{i+1}oo(1o)*[1]\}$. If $f(b_1b_2..b_{i+1}oo1(o1)*[o]) = f(b_2..b_{i+1}oo1(o1)*[o]) = c_2..c_{i+1}oo(1o)*[1]$, then one easily verifies again that claim 5.3.2.5.1. is contradicted. Thus $f(b_1..b_{i+1}oo1(o1)*[o]) = c_1c_2..c_{i+1}oo(1o)*[1]$, for some $c_1 \in \frac{o}{1}$. This completes the inductive argument. We conclude in particular (take $i=n-3$) that for every $b_1..b_{n-3} \in (\frac{o}{1})^{n-3}$ there exists a $c_1..c_{n-3} \in (\frac{o}{1})^{n-3}$ such that $f(b_1..b_{n-3}oo1) = c_1..c_{n-3}oo$.

Next consider $f(b_1..b_{n-3}ooo)$. Since $f(b_2..b_{n-3}ooo1) = c_1..c_{n-3}oo$ for suitable $c_1..c_{n-3} \in (\frac{o}{1})^{n-3}$, it follows that $f(b_1b_2..b_{n-3}ooo) \in \{c_1..c_{n-3}oo, \frac{o}{1}c_1..c_{n-3}o\}$. If $b_1..b_{n-3} = o^{n-3}$, then necessarily $f(b_1..b_{n-3}ooo) = o^{n-1}$ by claim 5.3.2.5.4. and the form claimed under (i) holds. Thus let $b_1..b_{n-3} \ne o^{n-3}$. If $f(b_1..b_{n-3}ooo) = c_1..c_{n-3}oo$, then the form claimed under (i) holds too. Hence let $f(b_1..b_{n-3}ooo) = \beta c_1..c_{n-3}o$ and, consequently, $f(b_ob_1..b_{n-3}oo) \in \{\beta c_1..c_{n-3}o, \frac{o}{1}\beta c_1..c_{n-3}\}$ for some $\beta \in \frac{o}{1}$ and $b_o \in \frac{o}{1}$. (Note that necessarily $b_ob_1..b_{n-3}oo \ne b_1..b_{n-3}ooo$.) If $f(b_o..b_{n-3}oo) = f(b_1..b_{n-3}ooo) = \beta c_1..c_{n-3}o$, then it follows from claim 5.3.2.5.1. that $\beta c_1..c_{n-3}o \in \{o^{n-1}, (o1)*[o], (1o)*[1]\}$. In each of the three cases uniformity is contradicted. (Note that $o^n \in f^{-1}(o^{n-1})$ by claim 5.3.2.5.4. in this case, and $(o1)*[o] \in f^{-1}((o1)*[o])$ and $(1o)*[1] \in f^{-1}((1o)*[1])$.) Thus $f(b_o..b_{n-3}oo) =$

$\frac{o}{1}\beta c_1..c_{n-3}$. Since $f(b_1..b_{n-3}oo1) = c'_1..c'_{n-3}oo$ for suitable $c'_1..c'_{n-3} \in (\frac{o}{1})^{n-3}$, it follows that $f(b_ob_1..b_{-3}oo) \in \{c'_1..c'_{n-3}oo, \frac{o}{1}c'_1..c'_{n-3}o\}$ and thus ends with a "o". Hence $c_{n-3} = o$, and $f(b_1..b_{n-3}ooo) = \frac{o}{1}c_1..c_{n-4}oo$ as claimed. □

We now begin our case analysis.

Claim 5.3.2.5.6. For n≥4, the case f(oo(1o)*[1]) = oo(1o)*[1] and f(11(o1)*[o]) = 11(o1)*[o] is contradictory.

Proof.

By claim 5.3.2.5.5. the 2^{n-2} strings of $(\frac{o}{1})^{n-3}ooo \cup (\frac{o}{1})^{n-3}oo1$ are mapped to the 2^{n-3} strings of $(\frac{o}{1})^{n-3}oo$. By uniformity it follows that no other strings can be mapped to $(\frac{o}{1})^{n-3}oo$. Likewise no other strings than the elements of $(\frac{o}{1})^{n-3}11o \cup (\frac{o}{1})^{n-3}111$ are mapped to $(\frac{o}{1})^{n-3}11$. Let $b_1..b_{n-3} \in (\frac{o}{1})^{n-3}$. By claim 5.3.2.5.5. we have $f(b_2..b_{n-3}o11o) = c_1..c_{n-3}11$ and $f(b_2..b_{n-3}1ooo) = c'_1..c'_{n-3}oo$ and, consequently, $f(b_1..b_{n-3}o11) \in \{c_1..c_{n-3}11, \frac{o}{1}c_1..c_{n-3}1\}$ and $f(b_1..b_{n-3}1oo) \in \{c'_1..c'_{n-3}oo, \frac{o}{1}c'_1..c'_{n-3}o\}$. The cases that $f(b_1..b_{n-3}o11) = f(b_2..b_{n-3}o11o) = c_1..c_{n-3}11$ or $f(b_1..b_{n-3}1oo) = f(b_2..b_{n-3}1ooo) = c'_1..c'_{n-3}oo$ clash with claim 5.3.2.5.1. Thus $f(b_1..b_{n-3}o11) = \frac{o}{1}c_1..c_{n-3}1$ and $f(b_1..b_{n-3}1oo) = \frac{o}{1}c'_1..c'_{n-3}o$ and, since neither one can "end" with oo or 11, we have in fact that $f(b_1..b_{n-3}o11) = d_1..d_{n-3}o1$ and $f(b_1..b_{n-3}1oo) = d'_1..d'_{n-3}1o$ (for suitable $d_1..d_{n-3}$ and $d'_1..d'_{n-3}$). By a very similar argument one now shows that $f(b_1..b_{n-3}o1o) = e_1..e_{n-3}o1$ and $f(b_1..b_{n-3}1o1) = e'_1..e'_{n-3}1o$, for suitable $e_1..e_{n-3}$ and $e'_1..e'_{n-3}$. (It follows that f 'resembles' f_1 of table A.)

As f was assumed not to be step-simulating, there must be $x \in (\frac{o}{1})^{n-1}$ and $y \in (\frac{o}{1})^{n-3}$ and $\alpha,\beta,\gamma,\delta \in (\frac{o}{1})$ such that $f(\alpha x) = f(x\gamma) = \beta y\delta$ and $\beta y \neq y\delta$. By claim 5.3.2.5.1. one of the strings αx, $x\gamma$ is o^n or 1^n and hence (by claim 5.3.2.5.4.) $\beta y\delta \in \{o^n,1^n\}$, or $\beta y\delta \in$ {(o1)*[o], (1o)*[1]}. In the former case the condition $\beta y \neq y\delta$ is violated. In the latter case we necessarily have $\beta y\delta = \beta y'\overline{\delta}\delta$ (for a suitable y') and hence, by our earlier analysis, necessarily $\alpha x = \alpha x'\overline{\delta}\delta\frac{o}{1}$ for suitable x'. It follows that $x\gamma = x'\overline{\delta}\delta\frac{o}{1}\gamma$ and thus $f(x\gamma)$ ends in $\delta\frac{o}{1}$, contradicting that it equals $\beta y\delta$ and thus ends in $\bar{\delta}\delta$. □

Next let $f(oo(1o)^*[1]) = (o1)^*[o]$ and $f(11(o1)^*[o]) = 11(o1)^*[o]$. Observe that $f(oo(1o)^*[1]) = f((o1)^*[o]) = (o1)^*[o]$ and thus by uniformity, $f(1oo(1o)^*[1]) = \frac{o}{1}(o1)^*[o]$.

Claim 5.3.2.5.7. For $n \geq 4$, the case $f(oo(1o)^*[1]) = (o1)^*[o]$ and $f(11(o1)^*[o]) = 11(o1)^*[o]$ is contradictory.

Proof.

We distinguish two further cases.

(a) $f(1oo(1o)^*[1]) = o(o1)^*[o]$. As in the proof of claim 5.3.2.5.5. one shows by induction that for all $1 \leq i \leq n-3$ and $b_1..b_i \in (\frac{o}{1})^i$ there exists $c_1..c_i \in (\frac{o}{1})^i$ such that $f(b_1..b_i 1oo(1o)^*[1]) = c_1..c_i o(o1)^*[o]$. Thus for $b_1..b_{n-3} \in (\frac{o}{1})^{n-3}$ we have $f(b_1..b_{n-3}1oo) = c_1..c_{n-3}oo$. It also follows that for every $b_o \in (\frac{o}{1})$ $f(b_o b_1..b_{n-3}1o) = \frac{o}{1}c_1..c_{n-3}o$. For $b_{n-3}=1$ this contradicts claim 5.3.2.5.5. (ii).

(b) $f(1oo(1o)^*[1]) = 1(o1)^*[o] = f((1o)^*[1])$. By uniformity one must have $f(o1oo(1o)^*[1]) = 11(o1)^*[o]$. By induction one shows that for all $1 \leq i \leq n-4$ and $b_1..b_i \in (\frac{o}{1})^i$ there exists $c_1..c_i \in (\frac{o}{1})^i$ with $f(b_1..b_i o1oo(1o)^*[1]) = c_1..c_i 11(o1)^*[o]$. Thus for $i=n-4$ we have $f(b_1..b_{n-4}o1oo) = c_1..c_{n-4}11o$, and it follows that also for every $b_o \in \frac{o}{1}$ that $f(b_o b_1..b_{n-4}o1o) = \frac{o}{1}c_1..c_{n-4}11$. (For if $f(b_o b_1..b_{n-4}o1o) = f(b_1..b_{n-4}o1oo) = c_1..c_{n-4}11o$, one easily derives a contradiction with claim 5.3.2.5.1.) By claim 5.3.2.5.5. and a uniformity argument (cf. the proof of claim 5.3.2.5.6.), no other strings than the elements of $(\frac{o}{1})^{n-3}11o \cup (\frac{o}{1})^{n-3}111$ can be mapped to $(\frac{o}{1})^{n-3}11$. This contradicts the assertion for $f(b_o b_1..b_{n-4}o1o)$. □

By a similar argument the following cases are proved contradictory as well: $f(oo(1o)^*[1]) = (1o)^*[1]$ and $f(11(o1)^*[o]) = 11(o1)^*[o]$, $f(oo(1o)^*[1]) = oo(1o)^*[1]$ and $f(11(o1)^*[o]) = (o1)^*[o]$, and $f(oo(1o)^*[1]) = oo(1o)^*[1]$ and $f(11(o1)^*[o]) = (1o)^*[1]$.

Claim 5.3.2.5.8. For $n \geq 4$, the case $f(oo(1o)^*[1]) = (o1)^*[o]$ and $f(11(o1)^*[o]) = (1o)^*[1]$ is contradictory.

Proof.

By uniformity (recall that f((o1)*[o]) = (o1)*[o] and f((1o)*[1]) = (1o)*[1]) we necessarily have f([1](o1)*oo) = [1](o1)*oo, and also f([o](1o)*11) = [o](1o)*11. Thus we have a situation similar to the one considered in claim 5.3.2.5.5. and 5.3.2.5.6., with the orientation of the strings involved "reversed". Clearly a contradiction is again derived. □

The case f(oo(1o)*[1]) = (1o)*[1] and f(11(o1)*[o]) = (o1)*[o] is proved contradictory in the same way. By noting that the cases f(oo(1o)*[1]) = f(11(o1)*[o]) = (o1)*[o] and f(oo(1o)*[1]) = f(11(o1)*[o]) = (1o)*[1] cannot occur because of uniformity, the case analysis is complete.

Case V. f((01)*[o]) = (1o)*[1], f((1o)*[1]) = (o1)*[o].

This case is "dual" to case IV, which was shown to be contradictory.

Case VI. f((o1)*[o]) = f((1o)*[1]) = (1o)*[1].

This case is "dual" to case III, which was shown to be contradictory.

This completes the proof of theorem 5.3.2.5. □

Appendix C. The proof of the topological reduction theorem for emulations of C_n on C_{n-1} (theorem 5.5.5.)

We use the notations and terminology of Section 5.5. Our aim is to prove the following result.

Theorem 5.5.5. (Topological Reduction Theorem). Let $n \geq 4$, and let f be a uniform emulation of C_n on C_{n-1}. Then there exists an (n−1)-face A of C_n such that f(A) is an (n−2)-face of C_{n-1}.

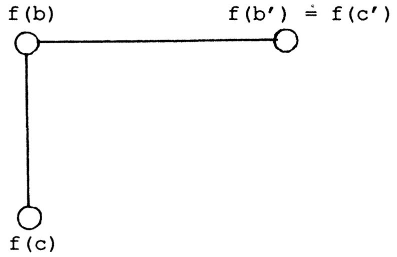
The proof proceeds by way of contradiction. Let $n \geq 4$, and let f be a uniform emulation of C_n on C_{n-1}. Suppose that there does not exist an (n−1)-face A of C_n such that f(A) is an (n−2)-face of C_{n-1}.

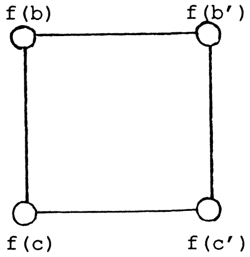
Claim 5.5.5.1. For every k with $1 \leq k \leq n-1$, there does not exist a k-face A of C_n such that f(A) is a (k−1)-face of C_{n-1}.

Proof.

Without loss of generality let k<n−1. Suppose the claim is false. Let k be the largest integer $\in$ 1..n−2 for which there exists a k-face A of C_n such that f(A) is a (k−1)-face of C_{n-1}. Without loss of generality we may assume that the elements of A have identical bits in the last n−k positions, hence $A = \{x\alpha u | x \in (\frac{o}{1})^k\}$ for certain $\alpha \in (\frac{o}{1})$ and $u \in (\frac{o}{1})^{n-k-1}$. Consider the (k+1)-face $A' = \{x\frac{o}{1}u | x \in (\frac{o}{1})^k\} = \{x\alpha u | x \in (\frac{o}{1})^k\} \cup \{x\alpha u | x \in (\frac{o}{1})^k\}$. For every $b = x\alpha u \in A$, let $b' = x\alpha u$. Because of uniformity no elements b′ can be mapped into f(A). It follows that f(b′) is obtained from f(b) by flipping one bit. We claim that for all $b,c \in A$ one has f(b)−f(b′) = f(c)−f(c′), where "−" denotes the component-wise subtraction, i.e., $(b_1..b_k)-(c_1..c_k) = (b_1-c_1..b_k-c_k) \in \{-1,o,1\}^k$. It is sufficient to prove this for pairs $b,c \in A$ with d(b,c) = 1. Note that f(b′), f(c′) $\notin$ f(A). Suppose f(b′) = f(c′). If f(b) = f(c), then f(b)\-f(b′) = f(c)−f(c′) and we are finished. If f(b) ≠ f(c), then necessarily d(f(b),f(c)) = 1 and (f(b),f(c)) is an edge of C_{n-1} (in fact, of f(A)). However, both f(b) and f(c) are connected to f(b′) = f(c′) $\notin$ f(A) too. It follows that C_{n-1} contains a triangle, which is impossible. Next suppose f(b′) ≠ f(c′). If f(b) = f(c), then one easily argues again that C_{n-1} contains a triangle, and a contradiction arises. If f(b) ≠ f(c), then the nodes must form a 4-cycle and hence (necessarily) a 2-face of

C_{n-1}. It follows that f(b)−f(b′) = f(c)−f(c′).





Using the claim we now argue that f(A′) is a k-face of C_{n-1}. Note that A′ = A ∪ {b′ | b ∈ A}. Fix a b ∈ A and assume that f(b′) is obtained from f(b) by flipping the i^{th} bit, where i belongs to the bit-positions with fixed values for face f(A). For arbitrary c ∈ A, the identity f(b)−f(b′) = f(c)−f(c′) forces that f(c′) is obtained from f(c) by flipping exactly the same i^{th} bit. Thus f(A′) ⊃ f(A) is a k-face of C_{n-1}. This contradicts that k was the largest integer for which a face of C_n with this property exists. □

We shall now prove a number of results that will eventually contradict claim 5.5.5.1., which thus proves that our initial assumption was false.

Definition. For 1≤k≤n−1, a k-face A of C_n is called stable if f(A) is a k-face of C_{n-1}.

Claim 5.5.5.2. There exists a 2-face A of C_n that is stable.

Proof.

Consider the 2-face A = {xoo..o | x ∈ $\binom{o}{1}^2$}. Suppose A is not stable, i.e., f(A) is not a 2-face of C_{n-1}. By uniformity f(A) contains at least 2 elements, but by claim 5.5.5.1. it can not be a 1-face. It follows that f(A) contains precisely 3 elements. Observing adjacencies, the following two cases can arise:

(a) f(oooo..o) = f(11oo..o) and f(o1oo..o) ≠ f(1ooo..o). Consider f(oo1o..o). By uniformity it cannot be equal to f(oooo..o) and f(11oo..o). If f(oo1o..o) = f(o1oo..o), then either f(1o1o..o) =

f(1ooo..o) or f(1o1o..o) is different from f(oooo..o), f(oo1o..o), and f(1ooo..o). In the former case f(oooo..o), f(o1oo..o) and f(1ooo..o) will form a triangle, which is impossible in C_{n-1}. In the latter case one verifies that $B = \{\alpha o\beta o..o \mid \alpha,\beta \in \frac{o}{1}\}$ is a stable 2-face of C_n. If f(oo1o..oo) = f(1ooo..o), then a similar argument shows that $B'=\{o\alpha\beta o..o \mid \alpha,\beta \in \frac{o}{1}\}$ must be a stable 2-face again. If f(oo1o..o) ≠ f(o1oo..o) and ≠ f(1ooo..o), then observe the following. If f(1o1o..o) would coincide with either f(o1oo..o), f(1ooo..o), or f(oo1o..o), then triangles are formed in C_{n-1}. Contradiction. Thus f(1o1o..o) is different from all these, and one verifies again that B is a stable 2-face.

(b) f(oooo..o) ≠ f(11oo..o) and f(o1oo..o) = f(1ooo..o). Consider f(1o1o..o) and distinguish cases as under (a). Once again triangles in C_{n-1} are formed (contradiction), or $B = \{\alpha o\beta o..o \mid \alpha,\beta \in (\frac{o}{1})\}$ or $B' = \{o\alpha\beta o..o \mid \alpha,\beta \in (\frac{o}{1})\}$ is proved a stable 2-face of C_n.

The cases "f(oooo..o) = f(o1oo..o) and f(11oo..o) ≠ f(o1oo..o)" and alike cannot arise, because it would lead to 1-faces being mapped to o-faces (points), contradicting claim 5.5.5.1. □

The proof of claim 5.5.5.2. shows, in fact, that either $(\frac{o}{1})^2 o^{n-2}$, $(\frac{o}{1})o(\frac{o}{1})o^{n-3}$ or $o(\frac{o}{1})^2 o^{n-3}$ must be a stable 2-face of C_n.

Claim 5.5.5.3. For every k with $2 \le k \le n-2$, there exists a k-face A of C_n that is stable.

Proof.

We induct on k. The case k=2 follows by claim 5.5.5.2. Assume it holds up to some k with $2 \le k < n-2$. Let A be a stable k-face of C_n. Without loss of generality we can let $A = \{x\alpha u \mid x \in (\frac{o}{1})^k\}$ for some $\alpha \in (\frac{o}{1})$ and $u \in (\frac{o}{1})^{n-k-1}$. Let $A' = \{x\bar{\alpha}u \mid x \in (\frac{o}{1})^k\}$ (a k-face), and for every $b = x\alpha u \in A'$ let $b' = x\bar{\alpha}u \in A'$. We show that there must exist a stable (k+1)-face.

Suppose first that $f(A) \cap f(A') = \emptyset$. As in the proof of claim 5.5.5.1. one shows that for all $b,c \in A$: $f(b)-f(b') = f(c)-f(c')$. Now note that f(A) is a k-face of C_{n-1}. As in the proof of claim 5.5.5.1. one shows that for all $b \in A$ f(b') is obtained from f(b) by flipping the same bit (in a position with fixed value for the elements of f(A)). Thus f(A') is a k-face of C_{n-1} too, and one easily verifies that $A \cup A' =$

$\{yu | y \in (\frac{0}{1})^{k+1}\}$ is a stable (k+1)-face of C_n.

Suppose next that $f(A) \cap f(A') \neq \emptyset$. If $f(A) = f(A')$, then $A \cup A'$ is a (k+1)-face of C_n whose image is a k-face (namely, $f(A)$) of C_{n-1} and a contradiction with claim 5.5.5.1. arises. Thus $f(A) \neq f(A')$, and it easily follows that $b',c' \in A'$ must exist with $d(b',c') = 1$ and $f(b') \notin f(A)$ and $f(c') \in f(A)$. (We assume that b,c are the corresponding nodes in A.) Let $\underline{b} \neq c$ be any other node $\in A$ adjacent to b, and let $\underline{c} \in A$ be obtained from c by flipping the same bit (as the one flipped to obtain $\underline{b}$ from b). We now claim : (i) $f(b) = f(c')$, (ii) $f(\underline{b}') \notin f(A)$, and (iii) $f(\underline{c}') \in f(A)$. For the proof, observe the following.

(i) Suppose $f(b) \neq f(c')$, and consider $f(c)$. If $f(c) = f(c')$ then $f(b)$, $f(b')$, and $f(c)$ form a triangle in C_{n-1} (by observing adjacencies). Contradiction. If $f(c) \neq f(c')$, then note that also $f(b) \neq f(c)$ (because f is necessarily 1-1 as a mapping from k-face A onto k-face $f(A)$). Thus $f(b)$, $f(b')$, $f(c')$, and $f(c)$ form a 4-cycle, hence a 2-face of C_{n-1}. But with $f(b), f(c')$, and $f(c)$ belonging to $f(A)$ the entire 2-face must belong to C_{n-1}, hence $f(b') \in f(A)$. Contradiction. We conclude $f(b) = f(c')$.

(ii) Suppose $f(\underline{b}') \in f(A)$. By uniformity $f(\underline{b}') \neq f(b) = f(c')$. If $f(\underline{b}') = f(\underline{b})$ then $f(b)$, $f(b')$, and $f(\underline{b}')$ form a triangle in C_{n-1}. Contradiction. If $f(\underline{b}') \neq f(\underline{b})$ then $f(b), f(b'), f(\underline{b})$, and $f(\underline{b}')$ form a 4-cycle, hence a 2-face of C_{n-1} with three nodes in the k-face $f(A)$. It follows that also $f(b') \in f(A)$. Contradiction. We conclude $f(\underline{b}') \notin f(A)$.

(iii) Note that $f(\underline{c}) \neq f(\underline{b})$, (else a contradiction with claim 5.5.5.1. arises), so $f(\underline{c})$ is adjacent to $f(\underline{b})$, and $f(\underline{b})$ is adjacent to $f(b)=f(c')$. Hence the distance between $f(c')$ and $f(\underline{c})$ is 2. $f(\underline{c}')$ must be adjacent to $f(c') \in f(A)$ and $f(\underline{c}) \in f(A)$, hence $f(c') \in f(A)$.

From the claim we derive that $\underline{b}',\underline{c}'$ is a pair exactly like b',c' and the argument can be repeated. In this way we can let b' range over all of A', and obtain that $f(A')$ must be a k-face of C_{n-1} and $f(A) \cap f(A')$ is a (k-1)-face (because nodes are paired in adjacent couples with one mapped to $f(A) \cap f(A')$ and the other to $f(A') - f(A)$). Now consider two more k-faces A'', A''' adjacent (parallel) to A obtained, say, by flipping the first and second bit of u respectively. (Note that $|u| \geq 2$, because $k<n-2$.) Either $f(A) \cap f(A'') = \emptyset$ or $f(A) \cap f(A''') = \emptyset$ and we would be

finished by the first part of the proof, or both f(A) ∩ f(A") ≠ ∅ and f(A) ∩ f(A''') ≠ ∅. In the latter case one derives the same conclusion for f(A") and f(A''') as for f(A'). It follows that $f^{-1}(f(A))$ contains at least 2^{k-1} elements of each A',A", and A''', thus at least $2\frac{1}{2}.2^{k-1}$ elements in all. This contradicts the uniformity of f.

This completes the induction argument. □

We now derive a contradiction as follows. By claim 5.5.5.3. there exists a (n-2)-face A of C_n that is stable. Without loss of generality we can let A = {xoo | x ∈ $(\frac{o}{1})^{n-2}$}. Let A' = {x1o | x ∈ $(\frac{o}{1})^{n-2}$}, A" = {xo1 | x ∈ $(\frac{o}{1})^{n-2}$}, and A''' = {x11 | x ∈ $(\frac{o}{1})^{n-2}$}. From the proof of claim 5.5.5.3. one derives that A',A", and A''' must be stable (n-2)-faces of C_n as well, and that the f-images of adjacent (parallel) faces are either disjoint or intersect (pairwise) in a (n-3)-face. We distinguish the following cases for the pairwise intersections :

(a) f(A) ∩ f(A') is an (n-3)-face, f(A) ∩ f(A") is an (n-3)-face. If f(A') ∩ f(A''') = ∅ or f(A") ∩ f(A''') = ∅, then A' ∪ A''' = {x1$\frac{o}{1}$ | x ∈ $(\frac{o}{1})^{n-2}$} or A" ∪ A''' = {x$\frac{o}{1}$1 | x ∈ $(\frac{o}{1})^{n-2}$} is stable (n-1)-face (as is its one parallel face A ∪ A" or A ∪ A', resp.) and either f(A) and f(A") or f(A) and f(A') must be disjoint respectively. Contradiction. We conclude that f(A') ∩ f(A''') and f(A") ∩ f(A''') both are (n-3)-faces too, in this case. Let b = xoo ∈ A and c' = y1o ∈ A' be such that f(b) = f(c') ∈ f(A) ∩ f(A'). Without loss of generality let f(A) = $(\frac{o}{1})^{n-3}(\frac{o}{1})\beta$ and f(A') = $(\frac{o}{1})^{n-3}\alpha(\frac{o}{1})$. Because f|A and f|A' act like isomorphisms of C_{n-2} theorem 5.5.4. applies, and there must be literals l_i and l'_i corresponding to b_i (1≤i≤n-2) and permutations Π and Π' such that $f(b_1..b_{n-2}oo)$ = $l_{\Pi(1)}..l_{\Pi(n-3)}l_{\Pi(n-2)}\beta$ and $f(b_1..b_{n-2}1o)$ = $l'_{\Pi'(1)}..l'_{\Pi'(n-3)}\alpha l'_{\Pi'(n-2)}$. By letting the argument $b_1..b_{n-2}$ range over $(\frac{o}{1})^{n-2}$ and observing that $f(b_1..b_{n-2}oo)$ and $f(b_1..b_{n-2}oo)$ and $f(b_1..b_{n-2,}1o)$ must have distance ≤ 1, one easily concludes that Π=Π' and $l_{\Pi(i)}$ = $l_{\Pi(i)}$ for 1≤i≤n-3. If f(xoo) = f(y1o) then necessarily x=y or d(x,y) = 1. Now let b' = x1o ∈ A', b" = xo1 ∈ A", b''' = x11 ∈ A''', and let c = yoo ∈ A, c" = yo1 ∈ A", c''' = y11 ∈ A'''. If x=y, then one obtains that the 1-face of C_n spanned by b and c' is mapped to a o-face (a point), contradicting claim 5.5.5.1. for k=1. If d(x,y) = 1, then b and c are adjacent and likewise are their primed companions. By a similar analysis of $f(A') \cap f(A''')$ and alike, one shows that necessarily : $f(b') = f(c''')$, $f(b'') = f(c)$, and $f(b''') = f(c'')$. It follows that the 3-face of C_n spanned by $b,b',b'',b''',c,c',c'',c'''$ is mapped to a 2-face of C_{n-1}. (The case that more f-value coincide is excluded by uniformity.) This contradicts claim 5.5.5.1. for k=3.

(b) $f(A) \cap f(A')$ is an (n-3)-face, $f(A) \cap f(A'') = \emptyset$. If $f(A') \cap f(A''')$ is an (n-3)-face, then one can use the argument under case (a) and derive a contradiction. Thus let $f(A') \cap f(A''') = \emptyset$. It follows that both $A \cup A'' = \{xo\binom{o}{1} \mid x \in \binom{o}{1}^{n-2}\}$ and $A' \cup A''' = \{x1\binom{o}{1} \mid x \in \binom{o}{1}^{n-2}\}$ are stable (n-1)-faces, thus their images each span C_{n-1}. It follows that $f(A) \cap f(A''')$ and $f(A'') \cap f(A''')$ cannot be empty, and thus must be (n-3)-faces. Now a similar argument as given under case (a) applies to derive a contradiction.

(c) $f(A) \cap f(A') = \emptyset$, $f(A) \cap f(A'') = \emptyset$. We may assume that $f(A') \cap f(A''') = \emptyset$ and $f(A'') \cap f(A''') = \emptyset$, otherwise analyses similar to case (a) and case (b) apply. It follows that $f(A) = f(A''')$ and $f(A') = f(A'')$, and the sets are complementary (n-2)-faces of C_{n-1}. Consider $b=xoo \in A$, $b'=x1o \in A'$, $b'' = xo1 \in A''$, and $b''' = x11 \in A'''$. Note that there is exactly one node in the (n-2)-face $f(A)$ that is adjacent to $f(b') \notin f(A)$. Hence $f(b) = f(b''')$. With a similar argument one shows $f(b') = f(b'')$. It follows that the 2-face of C_n spanned by b,b',b'',b''' is mapped to a 1-face. Contradiction with claim 5.5.5.1.

This ends the proof of theorem 5.5.5. □

REFERENCES

[An80] Angluin, D., Local and global properties in networks of processors, Proc. 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 82-93.

[AFdR80] Apt, K.R., N. Francez and W.P. de Roever, A proof system for Communication Sequential Processes, ACM Trans. Progr. Lang. & Syst. 2 (1980) 359-385.

[BM65] Barton, D.E., and C.L. Mallows, Some aspects of the random sequence, Ann. Math. Stat. 36 (1965) 236-260. (Reference [BM65] is not cited in the text.)

[Ba77] Barwise, J., (ed.), Handbook of mathematical logic, North Holland Publ. Comp., New York, 1977.

[Be83] Berman, F., Parallel processing with limited resources, Proc. of the Conf. on Information Sciences and Systems, pp.675-679, The Johns Hopkins University, 1983.

[Bg76] Berge, C., Graphs and hypergraphs, North Holland, New York, 1976.

[Bi1874] Bienaymé, J., Sur une question de probabilités, Bull. Soc. Math.France 2 (1874) 153-154.

[Bö84] Böhm, A.P.W., Dataflow computation, Ph.D.Thesis, Dept. of Computer Science, University of Utrecht, Utrecht, 1984.

[BH83] Broomell, G. and J.R. Heath, Classification categories and historical development of circuit switching topologies, ACM Comp. Surveys 15 (1983) 95-133.

[Bu80] Burns, J.E., A formal model for message passing systems, Techn. Rep. 91, Computer Sci. Dept., Indiana University, Bloomington, IN., 1980.

[Ch52] Chandler, K.N., The distribution and frequency of record values, J. Roy. Stat. Soc., Series B, 14 (1952) 220-228.

[CR79] Chang, E., and R. Roberts, An improved algorithm for decentralized extrema-finding in circular configurations of processes, C. ACM 22 (1979) 281-283.

[Co71] Cook, S.A., The complexity of theorem-proving procedures, Proc.

3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.

[Co85] Cornafion, Distributed computing systems, communication, cooperation, consistency, Elseviers Science Publ., Amsterdam, 1985.

[DB62] David, F.N., and D.E. Barton, Combinatorial chance, Charles Griffin & Comp., London, 1962.

[dB46] de Bruijn, N.G., A combinational problem, Indag. Math. VIII (1946) 461-467.

[De74] Deo, N., Graph theory with applications to engineering and computer science, Prentice Hall, Inc., Englewood Cliffs, NJ., 1974.

[DKR82] Dolev, D., M. Klawe, and M. Rodeh, An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle, J. Algorithms 3 (1982) 245-260.

[Ev84] Everhardt, P., Average case behaviour of distributed extrema-finding algorithms, Report ACT-49/T-147, Coordinated Science Lab., University of Illinois at U.C., 1984.

[Fe45] Feller, W., The fundamental limit theorems in probability theory, Bull. Amer. Math. Soc. 51 (1945) 800-832.

[Fe68] Feller, W., An introduction to probability theory and its applications, Vol.1, J. Wiley & Sons, New York, 1968.

[FF82] Fishburn, J.P. and R.A. Finkel, Quotient networks, IEEE Trans. Comput. C-31 (1982) 288-295.

[Fl86] Flajolet, P., personal communication, 1986.

[Fy72] Flynn, M.J., Some computer organisations and their effectiveness, IEEE Trans. Comput. C-21 (1972) 948-960.

[FS54] Foster, F.G., and A. Stuart, Distribution-free tests in time-series based on the breaking of records, J. Roy. Stat. Soc., Series B, 16 (1954) 1-13.

[Fr82] Franklin, W.R., On an improved algorithm for decentralized extrema finding in circular configurations of processors, C.ACM 25 (1982) 336-337.

[FL84] Frederickson, G.N., and N.A. Lynch, The impact of synchronous communication on the problem of electing a leader in a ring, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 493-503.

[Ga78] Galambos, J., The asymptotic theory of extreme order statistics, J. Wiley & Sons, New York, 1978.

[GHS83] Gallager, R.G., P.A. Humblet, and P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Prog. Lang. and Syst. 5 (1983) 66-77.

[GJ79] Garey, M.R., and D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman, San Francisco, Calif., 1979.

[GJT76] Garey, M.R., D.S. Johnson and R.E. Tarjan, The planar Hamiltonian circuit problem is NP-complete, SIAM J. Comput. 5 (1976) 704-714.

[Ge81] Gelernter, D., A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks, IEEE Trans. Comput. 10 (1981) 709-715.

[GdR84] Gerth, R. and W.P. de Roever, A proof system for concurrent ADA programs, Science of Comp. Progr. 4 (1984) 159-204.

[GPS76] Gibbs, N.E., W.G. Poole and P.K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J. Numer. Anal. 13 (1976) 236-250.

[GS84] Gurari, E.M. and I.H. Sudborough, Improved dynamic programming algorithms for bandwidth minimization and the MinCut lineair arrangement problem, J. of Algorithms 5 (1984) 531-546.

[HW73] Haghighi-Talab, D., and C. Wright, On the distribution of records in a finite sequence of observations, with an application to a road traffic problem, J. Appl. Prob. 10 (1973) 556-571.

[HSSD84] Halpern, J.Y., B.B. Simons, H.R. Strong, and D. Dolev, Fault-tolerant clock synchronization, Proc. 3rd Annual ACM Conf. on Principles of Distributed Computing, 1984, pp. 89-102.

[Ha69] Harary, F., Graph theory, Addison Wesley Publ. Comp., Reading Mass., 1969.

[Hl85] Halsall, F., Introduction to data communication and computer networks, Addison-Wesley, Wokingham, England, 1985.

[He71] Herman, G.T., When is a sequential machine the realization of another, Math. Syst. Th. 5 (1971) 115-127.

[HS80] Hirschberg, D.S., and J.B. Sinclair, Decentralized extrema-finding in circular configurations of processors, C.ACM 23 (1980) 627-628.

[Ho78] Hoare, C.A.R., Communicating sequential processes, C. ACM 21 (1978) 934-941.

[IPS84] Itai, A., C.H. Papadimitriou and J.L. Szwarzfiter, Hamiltonian paths in grid graphs, SIAM J. Comput. 11 (1984) 676-684.

[J74] Johnson, D.S., Approximation algorithms for combinatorial problems, J. Comp. Syst. Sci. 9 (1974) 256-278.

[JK77] Johnson, N.L. and S. Kotz, Urn models and their applications, J. Wiley & Sons, New York, 1977.

[KS60] Kemeny, J.G. and J.L. Snell, Finite markov chains, D. van Nostrand Co., Princeton, New Jersey, 1960.

[Kn81] Knuth, D.E., Verification of link-level protocols, BIT 21 (1981) 31-36.

[KvL83] Kramer, M.R., and J. van Leeuwen, Systolic computation and VLSI, in: J.W. De Bakker and J. van Leeuwen (eds.), Foundations of Computer Science IV ,part 1, pp. 75-100, Mathematical Centre Tract 158, Mathematical Centre, Amsterdam, 1983.

[KRS81] Korach, E., D. Rotem, and N. Santoro, A probabilistic algorithm for decentralized extrema-finding in a circular configuration of processors, Res. Rep. CS-81-19, Dept. of Computer Science, Univ. of Waterloo, Waterloo, 1981.

[La78] Lamport, L., Time, clocks, and the ordering of events in a distributed system, C. ACM 21 (1978) 558-565.

[LSP82] Lamport, L., R. Shostak and M. Pease, The Byzantine generals problem, ACM Trans. Progr. Lang. & Syst. 4 (1982) 382-401.

[LL77] LeLann, G., Distributed systems-towards a formal approach, in: B. Gilchrist (ed.), Information Processing 77 (IFIP), North-Holland Publ. Comp., Amsterdam, 1977, pp. 155-160.

[LL81] LeLann, G., Motivations, objectives and characterizations of distributed systems, in: D.W. Davies et.al., Distributed systems - architectures and implementation, Lect. Notes in Comp.

Sc. vol 105, Springer Verlag, Heidelberg, 1981.

[LVW84] Leung, J. Y-T., O. Vornberger and J.D. Witthoff, On some variants of the bandwidth minimization problem, SIAM J. Comput. 13 (1984) 650-666.

[MC80] Mead, C.A. and L.A. Conway, Introduction to VLSI-systems, Addison-Wesley Publ. Comp., Reading, Mass., 1980.

[MS80] Merlin, P.M. and P.J. Schweitzer, Deadlock avoidance in store and forward networks - I : Store-and-forward deadlock, IEEE Trans. Comm. 28 (1980) 345-354.

[MSZ85] Moran, S., M. Shalom, and S. Zaks, An algorithm for distributed leader finding in bidirectional rings without common sense of direction (draft), 1985.

[Mu85] Mullender, S.J., Principles of distributed operating system design, Ph.D. Thesis, Free University, Amsterdam, 1985.

[NS80] Nassimi, D., and S. Sahni, An optimal routing algorithm for mesh-connected parallel computers, J.ACM 27 (1980) 6-29.

[Oc84] OCCAM programming manual, Prentice Hall, Englewood Cliffs, New Jersey, 1984.

[PKR82] Pachl, J., E. Korach, and D. Rotem, A technique for proving lowerbounds for distributed maximum-finding algorithms, Proc. 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 378-382. Revised version: Lowerbounds for distributed maximum-finding algorithms, J.ACM 31 (1984) 905-918.

[Pa84] Paddon, D.J. (ed.), Supercomputers and parallel computation (proceedings), Clarendon Press, Oxford, 1984.

[Pa80] Parker, D.S., Notes on shuffle/exchange-type switching networks, IEEE Trans. Comput. C-29 (1980) 213-222.

[Pe82] Peterson, G.L., An O(nlogn) unidirectional algorithm for the circular extrema problem, ACM Trans. Prog. Lang. & Syst. 4 (1982) 758-762.

[PV81] Preparata, F.P. and J. Vuillemin, The cube-connected cycles, A versatile network for parallel computation, C. ACM 24 (1981) 300-309.

[Re32] Reidemeister, K., Einführung in die Kombinatorische Topologie, Friedr. Vieweg & Sohn Akt. Ges., Braunschweig, 1932.

[RS82] Reif, J. and S. Spirakis, Real time allocation in distributed systems, Proc. 2nd Annual Conf. on Principles of Distributed Computing, 1982, pp. 84-94.

[Re85] Reischuk, R., A new solution for the Byzantine generals problem, Information & Control 64 (1985) 23-42.

[RND77] Reingold, E.M., J. Nievergelt, and N. Deo, Combinatorial algorithms: theory and practice, Prentice Hall, Inc., Englewood Cliffs, NJ, 1977.

[Rn62] Renyi, A., Egy megfigyeléssorozat kiemelkedö elemeiröl (On the extreme elements of observations), MTA III. Oszt. Közl. 12 (1962) 105-121.

[Sa80] Saxe, J.B. Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time, SIAM J. Algebraic and Discrete Methods 1 (1980) 363-369.

[SKR82] Santoro, N., E. Korach, and D. Rotem, Decentralized extrema-finding in circular configurations of processors: an improved algorithm, Congr. Numer. 34 (1982) 401-412.

[SvL85] Schoone, A.A. and J. van Leeuwen, Verification of balanced link-level protocols, Techn. Rep. RUU-CS-85-12, University of Utrecht, Utrecht, 1985.

[Sh84] Shyamasunder, R.K., A simple lifelock-free algorithm for packet switching, Science of Comp. Programming 4 (1984) 249-256.

[Sh85] Sharp, J.A., Data flow computing, Ellis Horwood, Chichester, England, 1985.

[St84] Stallings, W., Local networks, ACM Comp. Surveys 16 (1984) 3-41.

[St85] Stallings, W., Data and computer communications, Macmillan Publ. Comp., New York, 1985.

[St71] Stone, H.S., Parallel processing with the perfect shuffle, IEEE Trans. Comput. C-20 (1971) 153-161.

[TvL86] Tan, R.B. and J. van Leeuwen, General symmetric distributed termination detection, Tech. Rep. RUU-CS-86-2, University of Utrecht, Utrecht, 1986.

[Ta81] Tanenbaum, A.S., Computer networks, Prentice Hall, Inc., Englewood Cliffs, NJ., 1981.

[TU81] Toueg, S., and J.D. Ullman, Deadlock-free packet switching networks, SIAM J. Comput. 10 (1981) 594-611.

[To80] Toueg, S., Deadlock and lifelock-free packet switching networks, Proc. 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 94-99.

[Tu86] Turner, J.S., On the probable performance of heuristics for bandwidth minimization, SIAM J. Comput. 15 (1986) 561-580.

[Ul84] Ullman, J.D., Flux, sorting and super-computer organization for AI applications, J. Parallel and Distr. Comp. 1 (1984) 133-151.

[vV85] van der Vorst, H.A., Comparative Performance tests of Fortran codes on the Cray-1 and Cyber 205, in: J. van Leeuwen and J.K. Lenstra (eds.), Parallel computers and computations, CWI Syll. 9, Centre for Mathematics and Computer Science, Amsterdam, 1985, pp. 33-54.

[vL83] van Leeuwen, J., Distributed computing, in: J.W. de Bakker and J. van Leeuwen (eds.), Foundations of Comp. Science IV (part 1), Mathematical Centre Tract 158, Mathematical Centre, Amsterdam, 1983, pp. 1-34.

[vL85] van Leeuwen, J., Parallel computers and algorithms, in: J. van Leeuwen and J.K. Lenstra (eds.), Parallel computers and computations, CWI Syll. 9, Centre for Mathematics and Computer Science, Amsterdam, 1985, pp. 1-32.

[vLT85] van Leeuwen, J., and R.B. Tan, An improved upperbound for decentralized extrema-finding in bidirectional rings of processors, Techn. Rep. RUU-CS-85-23, University of Utrecht, Utrecht, 1985.

[Vi84] Vitányi, P.M.B., Distributed elections in an archimedian ring of processors, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 542-547.

[WS76] Wai-Hung Liu and A.H.Sherman, Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices, SIAM J. Numer. Anal. 13 (1976) 198-213.

[Wi81] Wittie, L.D., Communication structures for large networks of microcomputers, IEEE Trans. Comput. C-30 (1981) 264-272.

MATHEMATICAL CENTRE TRACTS

1 T. van der Walt. *Fixed and almost fixed points.* 1963.

2 A.R. Bloemena. *Sampling from a graph.* 1964.

3 G. de Leve. *Generalized Markovian decision processes, part I: model and method.* 1964.

4 G. de Leve. *Generalized Markovian decision processes, part II: probabilistic background.* 1964.

5 G. de Leve, H.C. Tijms, P.J. Weeda. *Generalized Markovian decision processes, applications.* 1970.

6 M.A. Maurice. *Compact ordered spaces.* 1964.

7 W.R. van Zwet. *Convex transformations of random variables.* 1964.

8 J.A. Zonneveld. *Automatic numerical integration.* 1964.

9 P.C. Baayen. *Universal morphisms.* 1964.

10 E.M. de Jager. *Applications of distributions in mathematical physics.* 1964.

11 A.B. Paalman-de Miranda. *Topological semigroups.* 1964.

12 J.A.Th.M. van Berckel, H. Brandt Corstius, R.J. Mokken, A. van Wijngaarden. *Formal properties of newspaper Dutch.* 1965.

13 H.A. Lauwerier. *Asymptotic expansions.* 1966, out of print; replaced by MCT 54.

14 H.A. Lauwerier. *Calculus of variations in mathematical physics.* 1966.

15 R. Doornbos. *Slippage tests.* 1966.

16 J.W. de Bakker. *Formal definition of programming languages with an application to the definition of ALGOL 60.* 1967.

17 R.P. van de Riet. *Formula manipulation in ALGOL 60, part 1.* 1968.

18 R.P. van de Riet. *Formula manipulation in ALGOL 60, part 2.* 1968.

19 J. van der Slot. *Some properties related to compactness.* 1968.

20 P.J. van der Houwen. *Finite difference methods for solving partial differential equations.* 1968.

21 E. Wattel. *The compactness operator in set theory and topology.* 1968.

22 T.J. Dekker. *ALGOL 60 procedures in numerical algebra, part 1.* 1968.

23 T.J. Dekker, W. Hoffmann. *ALGOL 60 procedures in numerical algebra, part 2.* 1968.

24 J.W. de Bakker. *Recursive procedures.* 1971.

25 E.R. Paërl. *Representations of the Lorentz group and projective geometry.* 1969.

26 European Meeting 1968. *Selected statistical papers, part I.* 1968.

27 European Meeting 1968. *Selected statistical papers, part II.* 1968.

28 J. Oosterhoff. *Combination of one-sided statistical tests.* 1969.

29 J. Verhoeff. *Error detecting decimal codes.* 1969.

30 H. Brandt Corstius. *Exercises in computational linguistics.* 1970.

31 W. Molenaar. *Approximations to the Poisson, binomial and hypergeometric distribution functions.* 1970.

32 L. de Haan. *On regular variation and its application to the weak convergence of sample extremes.* 1970.

33 F.W. Steutel. *Preservation of infinite divisibility under mixing and related topics.* 1970.

34 I. Juhász, A. Verbeek, N.S. Kroonenberg. *Cardinal functions in topology.* 1971.

35 M.H. van Emden. *An analysis of complexity.* 1971.

36 J. Grasman. *On the birth of boundary layers.* 1971.

37 J.W. de Bakker, G.A. Blaauw, A.J.W. Duijvestijn, E.W. Dijkstra, P.J. van der Houwen, G.A.M. Kamsteeg-Kemper, F.E.J. Kruseman Aretz, W.L. van der Poel, J.P. Schaap-Kruseman, M.V. Wilkes, G. Zoutendijk. *MC-25 Informatica Symposium.* 1971.

38 W.A. Verloren van Themaat. *Automatic analysis of Dutch compound words.* 1972.

39 H. Bavinck. *Jacobi series and approximation.* 1972.

40 H.C. Tijms. *Analysis of (s,S) inventory models.* 1972.

41 A. Verbeek. *Superextensions of topological spaces.* 1972.

42 W. Vervaat. *Success epochs in Bernoulli trials (with applications in number theory).* 1972.

43 F.H. Ruymgaart. *Asymptotic theory of rank tests for independence.* 1973.

44 H. Bart. *Meromorphic operator valued functions.* 1973.

45 A.A. Balkema. *Monotone transformations and limit laws.* 1973.

46 R.P. van de Riet. *ABC ALGOL, a portable language for formula manipulation systems, part 1: the language.* 1973.

47 R.P. van de Riet. *ABC ALGOL, a portable language for formula manipulation systems, part 2: the compiler.* 1973.

48 F.E.J. Kruseman Aretz, P.J.W. ten Hagen, H.L. Oudshoorn. *An ALGOL 60 compiler in ALGOL 60, text of the MC-compiler for the EL-X8.* 1973.

49 H. Kok. *Connected orderable spaces.* 1974.

50 A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.G.L.T. Meertens, R.G. Fisker (eds.). *Revised report on the algorithmic language ALGOL 68.* 1976.

51 A. Hordijk. *Dynamic programming and Markov potential theory.* 1974.

52 P.C. Baayen (ed.). *Topological structures.* 1974.

53 M.J. Faber. *Metrizability in generalized ordered spaces.* 1974.

54 H.A. Lauwerier. *Asymptotic analysis, part 1.* 1974.

55 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 1: theory of designs, finite geometry and coding theory.* 1974.

56 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 2: graph theory, foundations, partitions and combinatorial geometry.* 1974.

57 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 3: combinatorial group theory.* 1974.

58 W. Albers. *Asymptotic expansions and the deficiency concept in statistics.* 1975.

59 J.L. Mijnheer. *Sample path properties of stable processes.* 1975.

60 F. Göbel. *Queueing models involving buffers.* 1975.

63 J.W. de Bakker (ed.). *Foundations of computer science.* 1975.

64 W.J. de Schipper. *Symmetric closed categories.* 1975.

65 J. de Vries. *Topological transformation groups, 1: a categorical approach.* 1975.

66 H.G.J. Pijls. *Logically convex algebras in spectral theory and eigenfunction expansions.* 1976.

68 P.P.N. de Groen. *Singularly perturbed differential operators of second order.* 1976.

69 J.K. Lenstra. *Sequencing by enumerative methods.* 1977.

70 W.P. de Roever, Jr. *Recursive program schemes: semantics and proof theory.* 1976.

71 J.A.E.E. van Nunen. *Contracting Markov decision processes.* 1976.

72 J.K.M. Jansen. *Simple periodic and non-periodic Lamé functions and their applications in the theory of conical waveguides.* 1977.

73 D.M.R. Leivant. *Absoluteness of intuitionistic logic.* 1979.

74 H.J.J. te Riele. *A theoretical and computational study of generalized aliquot sequences.* 1976.

75 A.E. Brouwer. *Treelike spaces and related connected topological spaces.* 1977.

76 M. Rem. *Associons and the closure statement.* 1976.

77 W.C.M. Kallenberg. *Asymptotic optimality of likelihood ratio tests in exponential families.* 1978.

78 E. de Jonge, A.C.M. van Rooij. *Introduction to Riesz spaces.* 1977.

79 M.C.A. van Zuijlen. *Emperical distributions and rank statistics.* 1977.

80 P.W. Hemker. *A numerical study of stiff two-point boundary problems.* 1977.

81 K.R. Apt, J.W. de Bakker (eds.). *Foundations of computer science II, part 1.* 1976.

82 K.R. Apt, J.W. de Bakker (eds.). *Foundations of computer science II, part 2.* 1976.

83 L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH system.* 1979.

84 H.L.L. Busard. *The translation of the elements of Euclid from the Arabic into Latin by Hermann of Carinthia (?), books vii-xii.* 1977.

85 J. van Mill. *Supercompactness and Wallman spaces.* 1977.

86 S.G. van der Meulen, M. Veldhorst. *Torrix I, a programming system for operations on vectors and matrices over arbitrary fields and of variable size.* 1978.

88 A. Schrijver. *Matroids and linking systems.* 1977.

89 J.W. de Roever. *Complex Fourier transformation and analytic functionals with unbounded carriers.* 1978.

90 L.P.J. Groenewegen. *Characterization of optimal strategies in dynamic games.* 1981.

91 J.M. Geysel. *Transcendence in fields of positive characteristic.* 1979.

92 P.J. Weeda. *Finite generalized Markov programming.* 1979.

93 H.C. Tijms, J. Wessels (eds.). *Markov decision theory.* 1977.

94 A. Bijlsma. *Simultaneous approximations in transcendental number theory.* 1978.

95 K.M. van Hee. *Bayesian control of Markov chains.* 1978.

96 P.M.B. Vitányi. *Lindenmayer systems: structure, languages, and growth functions.* 1980.

97 A. Federgruen. *Markovian control problems; functional equations and algorithms.* 1984.

98 R. Geel. *Singular perturbations of hyperbolic type.* 1978.

99 J.K. Lenstra, A.H.G. Rinnooy Kan, P. van Emde Boas (eds.). *Interfaces between computer science and operations research.* 1978.

100 P.C. Baayen, D. van Dulst, J. Oosterhoff (eds.). *Proceedings bicentennial congress of the Wiskundig Genootschap, part 1.* 1979.

101 P.C. Baayen, D. van Dulst, J. Oosterhoff (eds.). *Proceedings bicentennial congress of the Wiskundig Genootschap, part 2.* 1979.

102 D. van Dulst. *Reflexive and superreflexive Banach spaces.* 1978.

103 K. van Harn. *Classifying infinitely divisible distributions by functional equations.* 1978.

104 J.M. van Wouwe. *Go-spaces and generalizations of metrizability.* 1979.

105 R. Helmers. *Edgeworth expansions for linear combinations of order statistics.* 1982.

106 A. Schrijver (ed.). *Packing and covering in combinatorics.* 1979.

107 C. den Heijer. *The numerical solution of nonlinear operator equations by imbedding methods.* 1979.

108 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science III, part 1.* 1979.

109 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science III, part 2.* 1979.

110 J.C. van Vliet. *ALGOL 68 transput, part I: historical review and discussion of the implementation model.* 1979.

111 J.C. van Vliet. *ALGOL 68 transput, part II: an implementation model.* 1979.

112 H.C.P. Berbee. *Random walks with stationary increments and renewal theory.* 1979.

113 T.A.B. Snijders. *Asymptotic optimality theory for testing problems with restricted alternatives.* 1979.

114 A.J.E.M. Janssen. *Application of the Wigner distribution to harmonic analysis of generalized stochastic processes.* 1979.

115 P.C. Baayen, J. van Mill (eds.). *Topological structures II, part 1.* 1979.

116 P.C. Baayen, J. van Mill (eds.). *Topological structures II, part 2.* 1979.

117 P.J.M. Kallenberg. *Branching processes with continuous state space.* 1979.

118 P. Groeneboom. *Large deviations and asymptotic efficiencies.* 1980.

119 F.J. Peters. *Sparse matrices and substructures, with a novel implementation of finite element algorithms.* 1980.

120 W.P.M. de Ruyter. *On the asymptotic analysis of large-scale ocean circulation.* 1980.

121 W.H. Haemers. *Eigenvalue techniques in design and graph theory.* 1980.

122 J.C.P. Bus. *Numerical solution of systems of nonlinear equations.* 1980.

123 I. Yuhász. *Cardinal functions in topology - ten years later.* 1980.

124 R.D. Gill. *Censoring and stochastic integrals.* 1980.

125 R. Eising. *2-D systems, an algebraic approach.* 1980.

126 G. van der Hoek. *Reduction methods in nonlinear programming.* 1980.

127 J.W. Klop. *Combinatory reduction systems.* 1980.

128 A.J.J. Talman. *Variable dimension fixed point algorithms and triangulations.* 1980.

129 G. van der Laan. *Simplicial fixed point algorithms.* 1980.

130 P.J.W. ten Hagen, T. Hagen, P. Klint, H. Noot, H.J. Sint, A.H. Veen. *ILP: intermediate language for pictures.* 1980.

131 R.J.R. Back. *Correctness preserving program refinements: proof theory and applications.* 1980.

132 H.M. Mulder. *The interval function of a graph.* 1980.

133 C.A.J. Klaassen. *Statistical performance of location estimators.* 1981.

134 J.C. van Vliet, H. Wupper (eds.). *Proceedings international conference on ALGOL 68.* 1981.

135 J.A.G. Groenendijk, T.M.V. Janssen, M.J.B. Stokhof (eds.). *Formal methods in the study of language, part I.* 1981.

136 J.A.G. Groenendijk, T.M.V. Janssen, M.J.B. Stokhof (eds.). *Formal methods in the study of language, part II.* 1981.

137 J. Telgen. *Redundancy and linear programs.* 1981.

138 H.A. Lauwerier. *Mathematical models of epidemics.* 1981.

139 J. van der Wal. *Stochastic dynamic programming, successive approximations and nearly optimal strategies for Markov decision processes and Markov games.* 1981.

140 J.H. van Geldrop. *A mathematical theory of pure exchange economies without the no-critical-point hypothesis.* 1981.

141 G.E. Welters. *Abel-Jacobi isogenies for certain types of Fano threefolds.* 1981.

142 H.R. Bennett, D.J. Lutzer (eds.). *Topology and order structures, part 1.* 1981.

143 J.M. Schumacher. *Dynamic feedback in finite- and infinite-dimensional linear systems.* 1981.

144 P. Eijgenraam. *The solution of initial value problems using interval arithmetic; formulation and analysis of an algorithm.* 1981.

145 A.J. Brentjes. *Multi-dimensional continued fraction algorithms.* 1981.

146 C.V.M. van der Mee. *Semigroup and factorization methods in transport theory.* 1981.

147 H.H. Tigelaar. *Identification and informative sample size.* 1982.

148 L.C.M. Kallenberg. *Linear programming and finite Markovian control problems.* 1983.

149 C.B. Huijsmans, M.A. Kaashoek, W.A.J. Luxemburg, W.K. Vietsch (eds.). *From A to Z, proceedings of a symposium in honour of A.C. Zaanen.* 1982.

150 M. Veldhorst. *An analysis of sparse matrix storage schemes.* 1982.

151 R.J.M.M. Does. *Higher order asymptotics for simple linear rank statistics.* 1982.

152 G.F. van der Hoeven. *Projections of lawless sequences.* 1982.

153 J.P.C. Blanc. *Application of the theory of boundary value problems in the analysis of a queueing model with paired services.* 1982.

154 H.W. Lenstra, Jr., R. Tijdeman (eds.). *Computational methods in number theory, part I.* 1982.

155 H.W. Lenstra, Jr., R. Tijdeman (eds.). *Computational methods in number theory, part II.* 1982.

156 P.M.G. Apers. *Query processing and data allocation in distributed database systems.* 1983.

157 H.A.W.M. Kneppers. *The covariant classification of two-dimensional smooth commutative formal groups over an algebraically closed field of positive characteristic.* 1983.

158 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science IV, distributed systems, part 1.* 1983.

159 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science IV, distributed systems, part 2.* 1983.

160 A. Rezus. *Abstract AUTOMATH.* 1983.

161 G.F. Helminck. *Eisenstein series on the metaplectic group, an algebraic approach.* 1983.

162 J.J. Dik. *Tests for preference.* 1983.

163 H. Schippers. *Multiple grid methods for equations of the second kind with applications in fluid mechanics.* 1983.

164 F.A. van der Duyn Schouten. *Markov decision processes with continuous time parameter.* 1983.

165 P.C.T. van der Hoeven. *On point processes.* 1983.

166 H.B.M. Jonkers. *Abstraction, specification and implementation techniques, with an application to garbage collection.* 1983.

167 W.H.M. Zijm. *Nonnegative matrices in dynamic programming.* 1983.

168 J.H. Evertse. *Upper bounds for the numbers of solutions of diophantine equations.* 1983.

169 H.R. Bennett, D.J. Lutzer (eds.). *Topology and order structures, part 2.* 1983.

CWI TRACTS

1 D.H.J. Epema. *Surfaces with canonical hyperplane sections.* 1984.

2 J.J. Dijkstra. *Fake topological Hilbert spaces and characterizations of dimension in terms of negligibility.* 1984.

3 A.J. van der Schaft. *System theoretic descriptions of physical systems.* 1984.

4 J. Koene. *Minimal cost flow in processing networks, a primal approach.* 1984.

5 B. Hoogenboom. *Intertwining functions on compact Lie groups.* 1984.

6 A.P.W. Böhm. *Dataflow computation.* 1984.

7 A. Blokhuis. *Few-distance sets.* 1984.

8 M.H. van Hoorn. *Algorithms and approximations for queueing systems.* 1984.

9 C.P.J. Koymans. *Models of the lambda calculus.* 1984.

10 C.G. van der Laan, N.M. Temme. *Calculation of special functions: the gamma function, the exponential integrals and error-like functions.* 1984.

11 N.M. van Dijk. *Controlled Markov processes; time-discretization.* 1984.

12 W.H. Hundsdorfer. *The numerical solution of nonlinear stiff initial value problems: an analysis of one step methods.* 1985.

13 D. Grune. *On the design of ALEPH.* 1985.

14 J.G.F. Thiemann. *Analytic spaces and dynamic programming: a measure theoretic approach.* 1985.

15 F.J. van der Linden. *Euclidean rings with two infinite primes.* 1985.

16 R.J.P. Groothuizen. *Mixed elliptic-hyperbolic partial differential operators: a case-study in Fourier integral operators.* 1985.

17 H.M.M. ten Eikelder. *Symmetries for dynamical and Hamiltonian systems.* 1985.

18 A.D.M. Kester. *Some large deviation results in statistics.* 1985.

19 T.M.V. Janssen. *Foundations and applications of Montague grammar, part 1: Philosophy, framework, computer science.* 1986.

20 B.F. Schriever. *Order dependence.* 1986.

21 D.P. van der Vecht. *Inequalities for stopped Brownian motion.* 1986.

22 J.C.S.P. van der Woude. *Topological dynamix.* 1986.

23 A.F. Monna. *Methods, concepts and ideas in mathematics: aspects of an evolution.* 1986.

24 J.C.M. Baeten. *Filters and ultrafilters over definable subsets of admissible ordinals.* 1986.

25 A.W.J. Kolen. *Tree network and planar rectilinear location theory.* 1986.

26 A.H. Veen. *The misconstrued semicolon: Reconciling imperative languages and dataflow machines.* 1986.

27 A.J.M. van Engelen. *Homogeneous zero-dimensional absolute Borel sets.* 1986.

28 T.M.V. Janssen. *Foundations and applications of Montague grammar, part 2: Applications to natural language.* 1986.

29 H.L. Trentelman. *Almost invariant subspaces and high gain feedback.* 1986.

30 A.G. de Kok. *Production-inventory control models: approximations and algorithms.* 1987.

31 E.E.M. van Berkum. *Optimal paired comparison designs for factorial experiments.* 1987.

32 J.H.J. Einmahl. *Multivariate empirical processes.* 1987.

33 O.J. Vrieze. *Stochastic games with finite state and action spaces.* 1987.

34 P.H.M. Kersten. *Infinitesimal symmetries: a computational approach.* 1987.

35 M.L. Eaton. *Lectures on topics in probability inequalities.* 1987.

36 A.H.P. van der Burgh, R.M.M. Mattheij (eds.). *Proceedings of the first international conference on industrial and applied mathematics (ICIAM 87).* 1987.

37 L. Stougie. *Design and analysis of algorithms for stochastic integer programming.* 1987.

38 J.B.G. Frenk. *On Banach algebras, renewal measures and regenerative processes.* 1987.

39 H.J.M. Peters, O.J. Vrieze (eds.). *Surveys in game theory and related topics.* 1987.

40 J.L. Geluk, L. de Haan. *Regular variation, extensions and Tauberian theorems.* 1987.

41 Sape J. Mullender (ed.). *The Amoeba distributed operating system: Selected papers 1984-1987.* 1987.

42 P.R.J. Asveld, A. Nijholt (eds.). *Essays on concepts, formalisms, and tools.* 1987.

43 H.L. Bodlaender. *Distributed computing: structure and complexity.* 1987.