HILDESHEIMER
INFORMATIK-
BERICHTE

ISSN 0941-3014

Günther Stiege

Periodicity: An Application of Digraphs to Markov Chains

24/95 (August 95)

UNIVERSITÄT
HILDESHEIM

Dieser Bericht ist
herausgegeben vom

Institut für
Betriebssysteme &
Rechnerverbund

Postfach 10 13 63
31113 Hildesheim

Errata

The errata list contains known errors and their corrections. With the exception of short footnotes indicating the presence of an error and giving a link to the errata list, the original text has not been modified.

No.	Error description and correction	Error detection
1	In the proof of proposition 4.1, page 10, the statement: *"For each vertex v put into the work list by* ***inworklist*** *holds: All its ascendents which are not also descendents are at earlier positions in the worklist."* is wrong, since the way to a descendent of v may be blocked by a marked vertex. However, proposition 4.1 remains valid because the above statement holds for the first vertex of a (proper or improper) component occurring in the work list. This can be proved, for instance, using the acyclic structure of the reduced digraph (see section 6) .	Holger Buck (July 1997)
2	In procedure ***pathmatrix*** (table 4, page 13) the variable v erroneously has two different meanings. It should denote as a global variable the root of the actual component whereas the vertex with which *pathmatrix* is called should be denoted differently. The Procedure must read correctly: $pathmatrix(newpathlength, u)$ *1* for (each child w of u) *2* {if ((u and w belong to the same component) $\wedge$ (the entry $(newpathlength, w)$ in the path matrix is NULL)) *3* { add the entry $(newpathlength, w)$ to the path matrix; *4* if (w equals v)	Holger Buck (July 1997)

b

Periodicity: An Application of Digraphs to Markov Chains[1]

Günther Stiege[2]
Universität Hildesheim

Abstract. Periodicity as known from Markov chain theory is defined for directed graphs and its properties derived by purely graph theoretic means. An $O(n^3)$-complexity algorithm using breadth first search to find the period of a finite strongly connected component is presented.

Kurzfassung. Periodizität, wie sie von Markovketten bekannt ist, wird für gerichtete Graphen definiert. Die Eigenschaften werden auf rein graphentheoretische Art hergeleitet. Es wird ein Algorithmus der Komplexität $O(n^3)$, der Breitensuche benutzt, zur Bestimmung der Periode einer endlichen starken Zusammenhangskomponente vorgestellt.

Categories and Subject Descriptors: E.1 [**Data Structures**]: Graphs; G.2.2 [**Graph Theory**]: Path and circuit problems; G.3 [**Probability and Statistics**]

General Terms: Algorithms, Theory

Additional Keywords and Phrases: Periodicity, Markov chain

Contents

[1]This report is available by www from http: //www.informatik.uni-hildesheim.de/FB4/Berichte/1995.html and by ftp from ftp.informatik.uni-hildesheim.de/pub/Reports/hib95-24.ps.gz

[2]Prof.Dr.Günther Stiege, Universität Hildesheim, Institut für Betriebssysteme und Rechnerverbund, Samelsonplatz 1, D-31141 Hildesheim, Germany · email stiege@informatik.uni-hildesheim.de

1 Introduction

Markov chains[3] are an important field of stochastic processes and find applications in various areas, among them economics and computer science.

The behaviour of a Markov chain is governed by the probability distribution of the initial states and the transition probabilities. In many cases, the long run behaviour of the Markov chain depends only on the latter. In any case, the transition probabilities determine completely the structure in the set of states (state structure). Normally, the transition probabilities are given as a *transition matrix*. An example is shown in figure 1. Another widely used representation of

	0	1	2	3	4	5	·	·	n	$n+1$	·	·
0	q	p	0	0	0	0	·	·	0	0	·	·
1	q	0	p	0	0	0	·	·	0	0	·	·
2	q	0	0	p	0	0	·	·	0	0	·	·
3	q	0	0	0	p	0	·	·	0	0	·	·
4	q	0	0	0	0	p	·	·	0	0	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·
$n-1$	q	0	0	0	0	0	·	·	p	0	·	·
n	q	0	0	0	0	0	·	·	0	p	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·	·

$0 < p, q < 1;\ p + q = 1$

Figure 1: *Transition Matrix of a Markov Chain*

transition probabilities is a *transition graph*. Figure 2 shows the transition graph corresponding

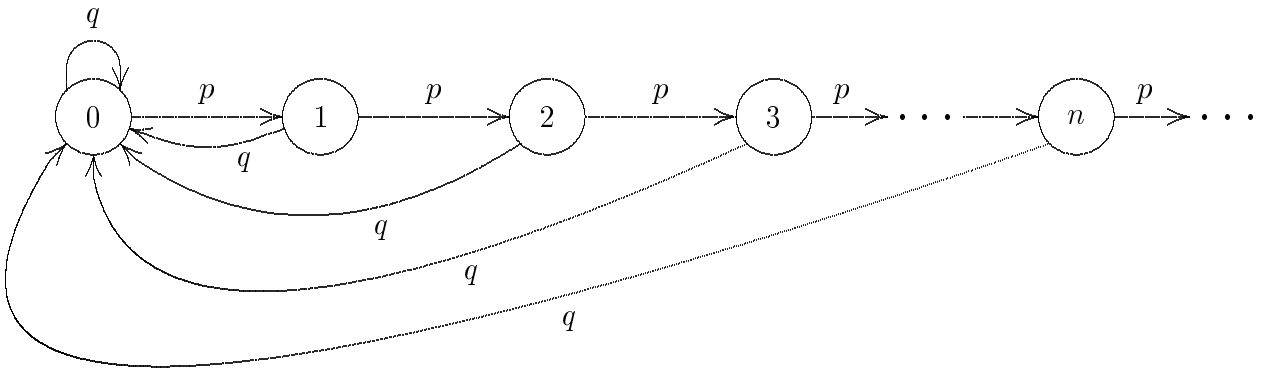

Figure 2: *Transition Graph of a Markov Chain*

[3]To be precise: Discrete time Markov chains with stationary transition probabilities.

to the transition matrix of figure 1. The transition graph of a Markov chain is a weighted directed graph (weighted digraph), with an arc from state a to state b iff[4] the transition probability from a to b is positive. In this case, the transition probability is the arc weight. The weight of a path[5] in the transition graph is the product of its arcs' weights and gives the (conditional) probability of exactly seeing the sequence of states given by the path, when started at the path's initial state.

The properties of the states of a Markov chain which determine the state structure and which are used to classify the states are connectivity, periodicity, and recurrence. In the case of Markov chains with finite state space, all three property classes depend only on the transition graph's structure and remain identical when the transition probabilities change, as long as there are no changes from positive to zero or vice versa.

In the case of Markov chains with infinite state space, this is true only for connectivity and periodicity. Recurrence properties (i. e. a state is either transient, null recurrent, or positive recurrent), do depend on the numerical values of the transition probabilities, i. e. on the weights of the arcs of the transition graph.

In any case, connectivity and periodicity of a Markov chain are important properties and can be defined and investigated by purely graph theoretic (and rather elementary) means.

Standard text books covering Markov chains sometimes do not use graph theoretic representations at all (Ross [Ross83], Feller [Fell68], Chung [Chun67], Langrock/Jahn [LaJa79], Resnick [Resn92], Asmussen [Asmu87], Rosenblatt [Rose74], Kohlas [Kohl77], Grimmet/Stirzaker [GrSt82], Gnedenko [Gned87], Mathar/Pfeifer [MaPf90], Kemeny/Snell [KeSn76], Freedman [Free83], Revuz [Revu84]) or use them as a help for intuition, mostly in examples (Çinlar [Cinl75], Isaacson/Madsen [IsMa76], Osaki [Osak92], Fahrmeir/Kaufmann/Ost [FaKO81]). In proofs and definitions, graph theoretic concepts and arguments are used, if at all, only in an indirect, hidden, and inconsistent manner. They are not clearly separated from probabilistic and analytical arguments.[6]

Authors of text books covering graph theory, always endeavour to present nice application examples, seem not to have noticed Markov chains. Particulary, they do not treat periodicity of digraphs at all (Aho/Hopcroft/Ullman [AhHU83] and [AhHU74], Carré [Carr79], Chen [Chen90], Gibbons [Gibb85], Bogart [Boga83] Böhm/Weise [BoWe81], Horowitz/Sahni [HoSa76], König [Koen90], Jungnickel [Jung90], Kohlas [Kohl87], Sachs [Sach70] Oberschelp/Wille [ObWi76], Sedgewick [Sedge91], Mehlhorn [Mehl84]). There are some exceptions: Anderson [Ande70] and Gondran/Minoux [GoMi84] contain some remarks on connectivity in finite Markov chains. Noltemeier [Nolt76] uses powers of the adjacency matrix of the transition graph to classify the states of a finite Markov chain and to calculate the period of a strongly connected component. Maurer/Ralston [MaRa91] offer both, a chapter on digraphs and a chapter on Markov chains. They claim to point out an interplay between Markov chains and digraphs, but use digraphs only to appeal to intuition.

This report attempts to fill the gap. The rest of the report is organized as follows: In section 2 connectivity is treated for the sake of completeness. All graph theoretic results are well known and standard text book material. A "Markov chain view" is pointed out. Section 3 introduces periodicity of digraphs and presents all important properties by purely graph theoretic means.

[4] if and only if

[5] Paths in digraphs are always assumed to be directed paths.

[6] An explicit and clear mention of graph theoretic methods is made in Medhi [Medh94]. A formula due to Solberg [Solb75] suitable to calculate the stationary probability distribution of a finite, aperiodic, and irreducible Markov chain is presented.

In section 4 an algorithm to find the structure of finite digraphs is presented. Depth first search is used in the well known manner to find the components of a digraph. Breadth first search is applied to find the periodicity of a component. In section 5 some efficiency considerations are given. Concluding remarks are contained in section 6.

Terminology: In the following, we shall speak of *vertices* or *nodes* of a digraph instead of states of a Markov chain. An *arc* points from a *parent (initial vertex, predecessor)* to a *child (terminal vertex, successor)*. A *path* is a finite (non-empty) sequence of arcs, where each arc's terminal vertex is the initial vertex of the next arc. The number of arcs of a path is the *path length*. A path is *closed* or a *cycle* iff the initial vertex of the first arc and the terminal vertex of the last arc are the same. A *loop* is a one-arc cycle. A path which is not closed is *simple* iff all its vertices are pairwise distinct. A cycle is *simple* iff it is a loop or iff removing its last arc a simple not closed path results. A vertex b is *accessible* (can be *reached*) from a vertex a iff there is a path from a to b, i. e. the initial vertex of the first arc is a and the terminal vertex of the last arc is b. Any vertex which can be reached from a is a *descendent* of a. Any vertex from which b is accessible is an *ascendent* of b. A *run* is a (non-empty) finite or infinite sequence of arcs for which each finite head is a path. Associated with a run is the idea of a token traveling through the digraph in discrete time *steps*, going in each step from one vertex to the next.

2 Connectivity

A vertex of a digraph is called *vertex of return* iff there is a path from the vertex to itself. Iff there is no such path the vertex is called a *vertex of no return*. Vertices H and D of the digraph of figure 3 are vertices of no return, all others are vertices of return.

As easily can be verified, *strong connectivity* (i. e. *mutual accessibility*) is an equivalence relation in the set of vertices of return of a digraph. Each equivalence class of mutual accessible vertices and also each vertex of no return is called a strongly connected component, *component* for short. In figure 3 dotted lines are used to show the components. A component consisting of vertices of return is called a *proper* component.

Vertices F and A of figure 3 are both vertices of return. However, there is an important difference between them. All vertices accessible from A belong to the same component, i. e. from all these vertices there is a path back to A. This is not the case for F, as for instance, D can be accessed from F. A vertex of return from which only vertices of the same component can be accessed is called a *vertex of return with no escape*. A vertex of return from which vertices outside its component can be accessed is called a *vertex of return with escape*. Of course, escapability is a component property: Either all vertices of a proper component are vertices of no escape or all are vertices of escape. Borrowing terminology from Markov chains, a vertex of return with no escape is called *essential*, the component to which it belongs is called *irreducible*, *closed*, or *final*. A vertex of no return or a vertex of return with escape is called *unessential*. The denominations stem from the fact that in a Markov chain all unessential states are transient and, if the state space is finite, all essential states are positive recurrent (see for instance [Cinl75], [IsMa76], [Ross83], or [Stie95c]). In the example of figure 3 the components generated by A and I are closed, the component generated by G is not closed.

The transition graph of a Markov chain has no terminal vertices. If it is finite, there exists at least one essential vertex as stated by the following proposition.

Proposition 2.1 *Each finite digraph with no terminal vertices contains an essential vertex.*

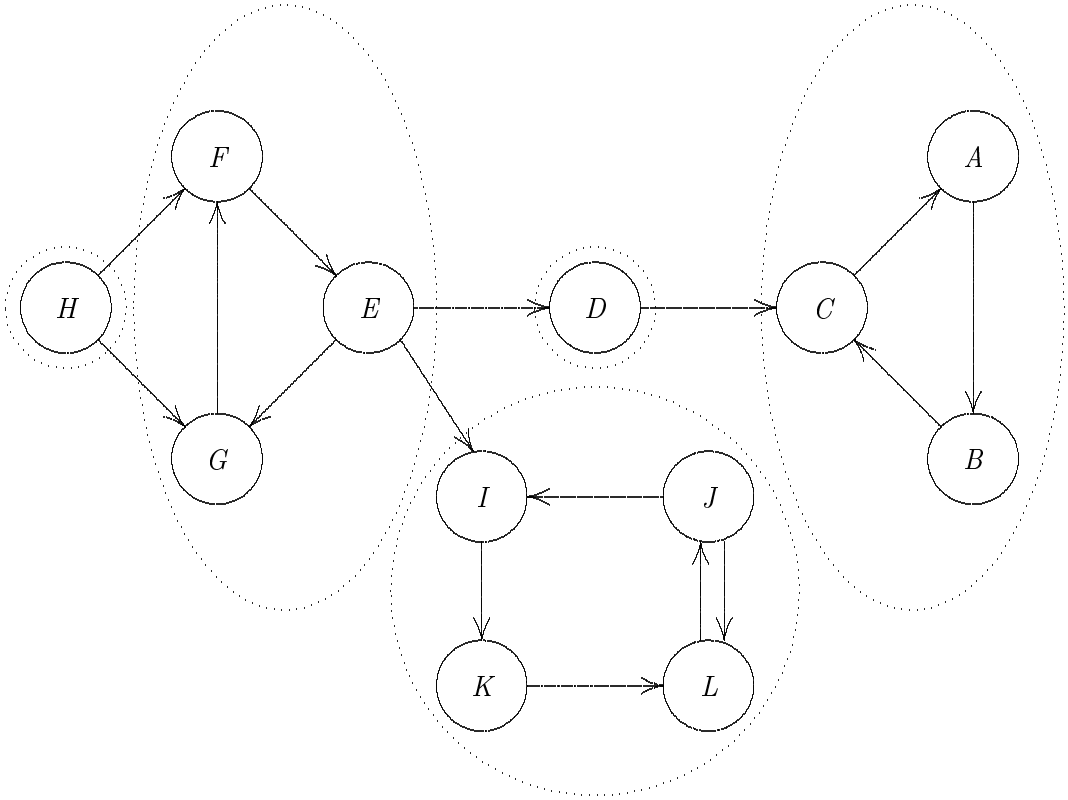

Figure 3: *Strongly Connected Components of a digraph*

Proof: The proposition is true if there is no unessential vertex. Be i_0 an unessential vertex. Then there exists i_1 accessible from i_0 but i_0 not accessible from i_1. In particular, i_0 is different from i_1. If i_1 is unessential, too, then there exists i_2 such that i_0 and i_1 are not accessible from i_2. Again, i_2 is different from i_0 and i_1. This construction is continued until a vertex is reached, which only has essential children. This must be the case, for otherwise an infinite sequence of pairwise distinct vertices could be constructed contradicting the finiteness of the digraph. $\Box$[7]

3 Periodicity

See figure 3. If a run is started with step 0 at vertex A, then A will be visited at step n, iff n is a multiple of 3. This property is important for Markov chains and gives rise to the following definition.

Definition 3.1 *A vertex of return i of a digraph has period p_i ($p_i \in \mathbf{N}$) iff the length of all paths from i to i is a multiple of p_i and there is no $p' > p_i$ with the same property.*

Obviously, every vertex of return i has a uniquely defined period which is given by

$$p_i := \gcd\{\text{lengthof}(w) \mid w \in P\},$$ [8]

where P is the set of all paths from i to i. If $p_i = 1$, i is called *aperiodic*. Periodicity is a component property:

[7] $\Box$ means end of a proof.

[8] gcd: *greatest common divisor*

Proposition 3.1 *Two mutually accessible vertices of a digraph have the same period.*

Proof: Be i and j mutually accessible vertices of a digraph and be p_i the period of i and p_j the period of j. We chose a closed path from i to j and back to i. Its length be l. This path, being also a path from j to j yields $l = k_1 \cdot p_j$. Then a second, completely arbitrary path from i to i is chosen (see figure 4). Its length be n. From the two paths a new path is constructed: From j

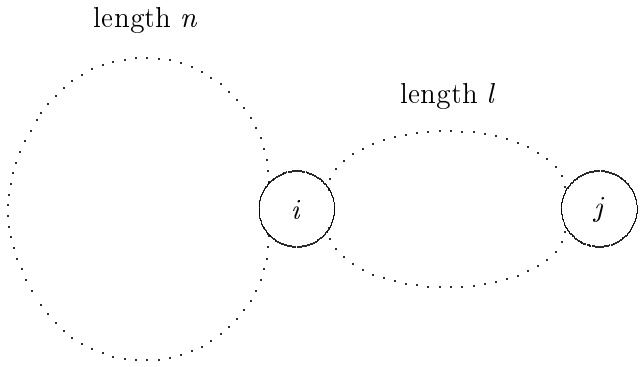

Figure 4: *Mutually accessible vertices have the same period*

to i following the first path, then from i to i following the second path and returning from i to j along the first path. The length of the compound path is $l + n$ and, for being a path from j to j there exists k_2 with $l + n = k_2 \cdot p_j$. Therefore

$$n = k_2 \cdot p_j - k_1 \cdot p_j = (k_2 - k_1) \cdot p_j.$$

The length of each path from i to i is a multiple of p_j, so $p_i \geq p_j$. In the same way $p_j \geq p_i$ can be shown, from which

$$p_i = p_j.$$

□

Checking all paths from i to i can be very difficult. So, other criteria to determine the period of a component are needed. In Markov chain theory *paths of first return* are important. A path of first return to vertex i is a path from i to i where i is visited only at the beginning and at the end. Be $\widetilde{P_i}$ the set of paths of first return to vertex i.

Proposition 3.2 *Be i a vertex of return of a digraph. Then*

$$p_i = \gcd\{\text{lengthof}(w) \,|\, w \in \widetilde{P}_i\}.$$

Proof: Be $A := \{\text{lengthof}(w) \,|\, w \in P_i\}$ and $B := \{\text{lengthof}(w) \,|\, w \in \widetilde{P}_i\}$. From $B \subseteq A$ follows $\gcd(B) \geq \gcd(A)$. If l divides all $n \in B$, l also divides all sums of numbers in B. As each path from i to i is a sequence of paths with first return to i, its length is the sum of the lengths of these and therefore divisible by l. That means $\gcd(B) \leq \gcd(A)$ from which

$$\gcd(B) = \gcd(A).$$

□

From Proposition 3.1 follows that the set of all cycles of a proper component has the period as greatest common divisor. Using analogous arguments as in the prove above, it can be seen that

the set of all simple cycles of a proper component also has the period of that component as greatest common divisor. Concerning simple cycles, this is formulated as the following proposition.

Proposition 3.3 *If C is the set of all simple cycles of a proper component of a digraph and p its period, then*

$$p = \gcd\{\text{lengthof}(w) \mid w \in C\}.$$

□

A path of first return needs not be a simple cycle and, in general, it is not true, that the lengths of the simple cycles through a fixed vertex have the period as gcd. The only simple cycle of the irreducible digraph of figure 5 containing vertex B has length 5, but the period is 1, due to the

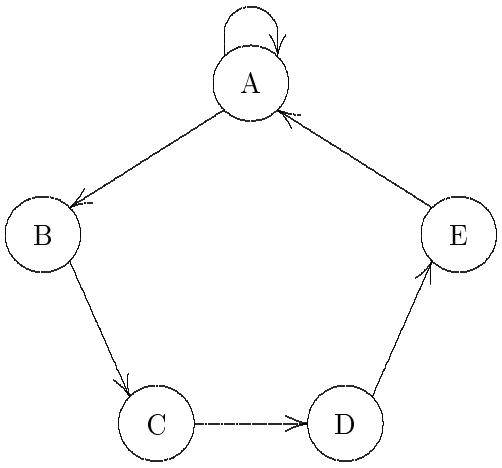

Figure 5: *Aperiodic component*

loop at vertex A. This example also shows that for all $n \geq 5$ there is a path of length n from B to B. The following proposition states that this property holds in general.

Proposition 3.4 *Be i an aperiodic vertex of a proper component of a digraph. Then there exists $n_0 \in \mathbf{N}$ such that for all $n \geq n_0$ there is a path from i to i of length n*

Proof: See for instance [IsMa76] or [Stie95c]. □

n_0 depends on i and cannot be smaller than the length of the shortest simple cycle through i. Therefore, in infinite components n_0 is in general unbounded in i.

If the period of a proper component is at least 2 that component can be structured further. As an example, see the digraph of figure 6. It consists of one closed component of period 2. From vertex A an even number of steps always leads back to A. Departing from a vertex B one arrives after an even number of steps always at a vertex B. The set of vertices is partitioned into two closed sets of vertices mutually accessible in 2 steps. The classes are

$$\{A\} \quad \text{and} \quad \{B_1, B_2, \cdots, B_n, \cdots\},$$

and with every single step a change from one class to the other is made. The following theorem states that this holds in general.

Theorem 3.1 *Be C a proper component of a digraph and be $p \geq 2$ its period. If only paths within the component are considered, then the following holds*

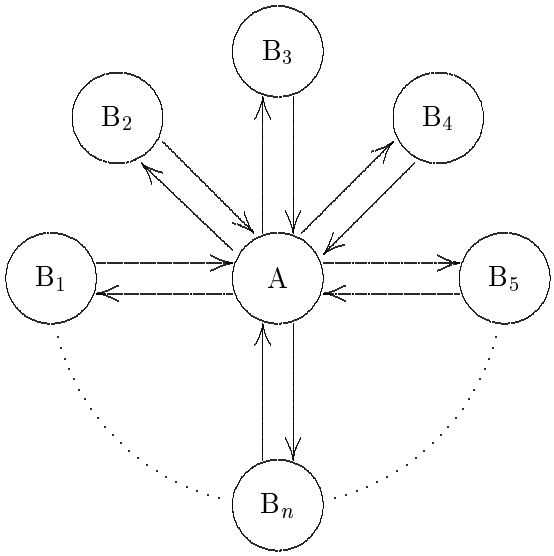

Figure 6: *Decomposition of a irreducible component of period 2*

1. *Mutual accessibility in $n \cdot p$ $(n \geq 1)$ steps is an equivalence relation over C.*

2. *C is partitioned into p equivalence classes. Each step leads from one class to another. Every p steps in sequence pass through all p classes in a fixed order and the last step ends in the first class of the sequence.*

3. *If only the vertices of one equivalence class are considered and a derived digraph structure of p-step-accessibility is imposed, this new digraph is closed and aperiodic.*

Proof:

1. Reflexivity is a consequence of periodicity p. Transitivity is obvious.
   Symmetry: Be there a path from i to j of length $n_1 \cdot p$. This path can be continued from j to i and the compound path is of length $n_2 \cdot p$, for p being the period. Thus there exists a path from j to i of length $(n_2 - n_1) \cdot p$.
   If remains to prove that *all* paths from one vertex to another vertex of the same class have length $n \cdot p$. This is the case, for otherwise a closed path of a length which is not a multiple of p could be constructed.

2. Be i_0 any vertex of the component and be given a closed path $i_0, i_1, \cdots, i_{p-1}, i_p, \cdots, i_0$. The vertices $i_0, i_1, \cdots, i_{p-1}$ belong to pairwise disjoint equivalence classes for otherwise a closed path with a length not being a multiple of p could be constructed.
   If j is an arbitrary vertex of C and a path from i_{p-1} to j is given, then choosing the right starting vertex from the vertices $i_0, \cdots, i_{p-1}$ there is always a path of length $n \cdot p$ from one of these vertices to j. So, there are exactly the p equivalence classes given by $i_0, i_1, \cdots, i_{p-1}$. Every sequence of steps visits the classes in the cyclic order

$$[i_0], [i_1], \cdots, [i_{p-1}], [i_0] \ .$$

$[i]$ is the class generated by i. In fact, be j a vertex accessible in one step from vertex i in $[i_n]$ $(0 \leq n \leq p-1)$. Let us choose a path

$$i_n, i_{(n+1) \bmod p}, \cdots, i, j.$$

The length of $i_n, i_{(n+1) \bmod p}, \cdots, i$ is a multiple of p, and so is the length of $i_{(n+1) \bmod p}, \cdots, i, j$. That means $j \in [i_{(n+1) \bmod p}]$.

3. The derived digraph is closed, for its vertices are mutually accessible and no other vertices are accessible. If the new digraph had period $f > 1$, then in the original equivalence class all closed paths had length $n \cdot f \cdot p$ and p were not the original period.

□

4 Algorithms

To effectively find the components and periodicity classes of a digraph, specific properties of that digraph can be used. In the case of infinite digraphs, this is the only way to establish the components and periodicity classes. In the case of finite digraphs there exist general algorithms, one of which will be presented in the following.

To be specific, let us assume the digraph is given as a data structure of the form of figure 7. The

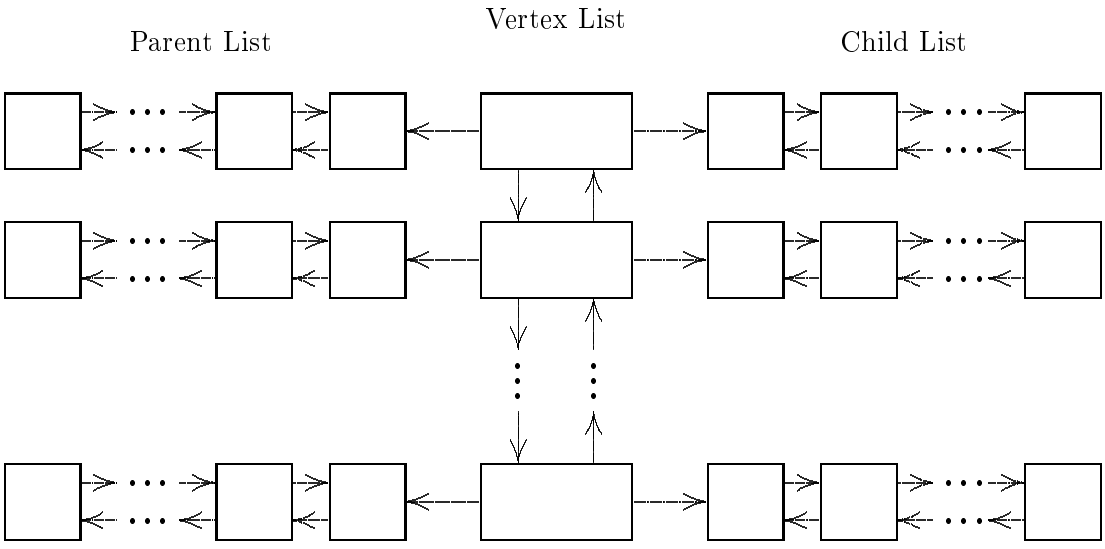

Figure 7: *Initial data structure representation of a digraph*

descriptions of the vertices, the vertex entries, form a linked list, the *vertex list*. Each vertex is head of two chains of arc entries, the *child list* and the *parent list*.

The data structure we want to obtain is shown in figure 8. A linked list of component entries is at the top *(component list)*. Each component entry points to a *periodicity pointer array* with number of entries equal to the period of that component, the order of the entries corresponding to the order of the periodicity classes (see theorem 3.1). If the component is a vertex of no return, there is 1 entry in the periodicity pointer array. Each entry in the periodicity pointer array is the head of the linked list of those vertices which are members of the corresponding periodicity class

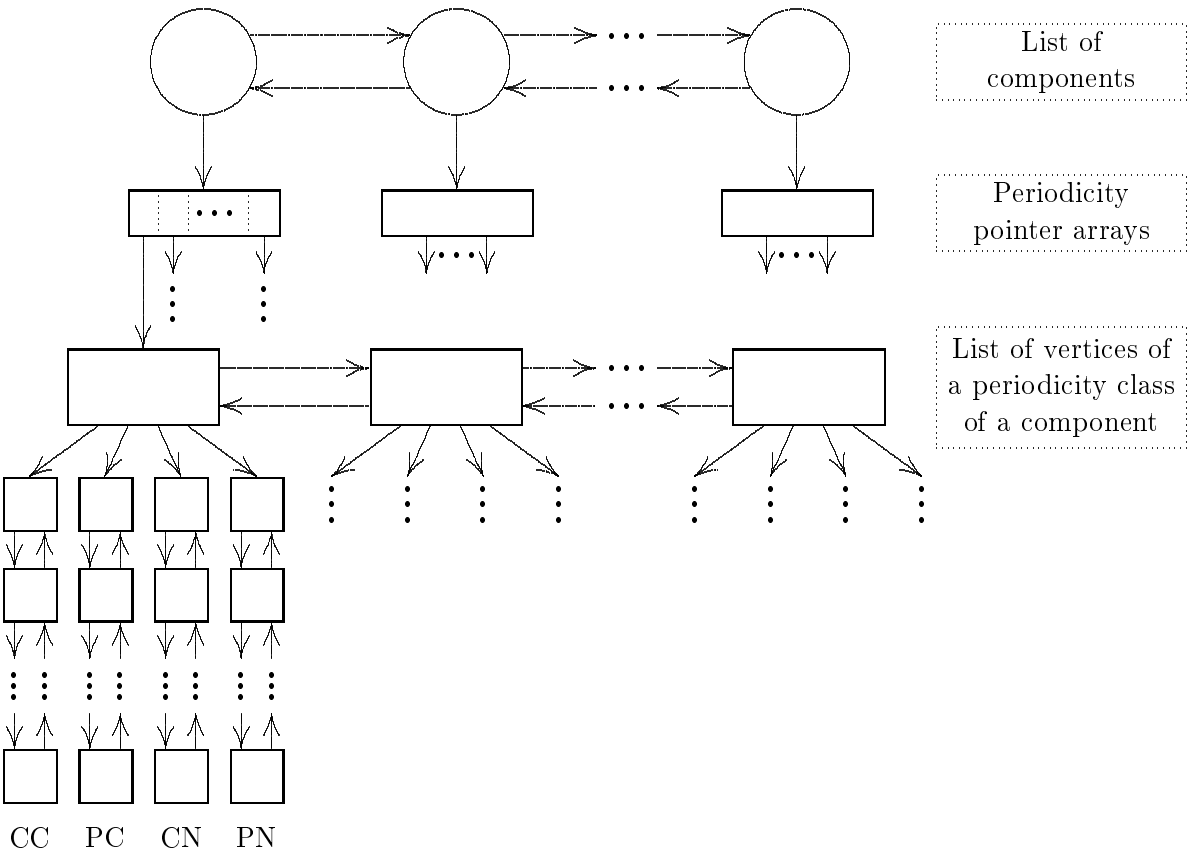

Figure 8: *Component/Periodicity representation of a digraph*

(periodicity list). Each vertex entry is head of 4 chains of arc entries: children in the class (CC), parents in the class (PC), children not in the class (CN), and parents not in the class (PN). We shall call this a *component/periodicity representation of a digraph.*

The algorithm presented in table 3 constructs the component/periodicity representation of a digraph from a representation as in figure 7. The algorithm is formulated in a C-like style.

Lines 1-3
Using depth first search, i. e. procedure *inworklist* (table 1), the vertices of the digraph are inserted into a work list in a LIFO manner. This is done in the order of procedure completion.

Lines 4-7:
Once the work list is complete, it is processed taking always the first element of the actual (i. e. remaining) work list as the root of a new component. The members of this component are found starting a depth first search in backward direction, procedure *buildmemberlist* (see table 2). All vertices found this way which are not yet members of a component are linked to the member list of the actual component. All these vertices are unlinked from the work list.

Lines 8-12:
If the member list is empty, the actual vertex on the work list, is a vertex of no return. The corresponding data structure is completed and the vertex removed from the work list.

It is well known that lines 1-12, especially procedure *inworklist* and *buildmemberlist* indeed yield the components of a digraph, see for instance [AhHU83]. For the sake of completeness, this

property is proved as proposition 4.1 below.

Lines 13-23:
If the member list is not empty, a component of mutually accessible vertices is given. To find its period and to establish its periodicity classes, a *pathmatrix* is used. This is a square matrix with path lengths 1 to *componentsize* as rows and vertex indices 0 to $componentsize - 1$ as columns. A non null entry (i, j) means that there is a path of length i from the component root to the vertex indexed j of the component. Procedure *pathmatrix* (see table 4) takes a path length and a vertex index as input and puts all children of the given vertex into the periodicity matrix row given by the path length. In addition, *pathmatrix* updates the value of *period*.
Lines 16-23 describe a breadth first search in the component, starting at the component root. The search is stopped when the period is found to be equal to 1 or if all paths from the component root with length up to and including *componentsize* are recorded in the periodicity matrix. Proposition 4.2 below proves that procedure *pathmatrix* indeed calculates correctly the period of the component.

Lines 24-27:
Once the period of the component is known, the periodicity pointer array is allocated. Using the entries in the periodicity matrix, the vertices of the component member list are distributed to the periodicity class lists using theorem 3.1.

Lines 28-36:
These statements partition the incoming and outgoing arcs of each vertex of the component into arcs within the component and arcs crossing components.

In the following the propositions announced are proved.

Proposition 4.1 *Lines 1-12 of the algorithm of table 3 including the procedures inworklist (table 1) and buildmemberlist (table 2) correctly find the strongly connected components of a digraph.*

Proof: For each vertex v put into the work list by *inworklist* holds:[9] All its ascendents which are not also descendents are at earlier positions in the worklist. In fact, such an ascendent is processed either in the same recursive sequence of *inworklist* and therefore completes its processing later than v or it is processed in a later recursive call sequence. In both cases the LIFO order of insertion into the worklist guarantees an earlier position. From this and the fact that all vertices already processed are removed from the worklist, it can be concluded by induction that the actual top of the worklist is root of a new component and all its ascendents not yet assigned to a component must also be accessible from it and so belong to the same component. □

Proposition 4.2 *Lines 15-23 of the algorithms of table 3 including procedure pathmatrix (table 4) correctly calculate the period of the component.*

Proof: Be v the root of the component with respect to which the periodicity matrix is built. According to proposition 3.3, the period of a component can be calculated as the greatest common divisor of the lengths of all simple cycles in that component. If a simple cycle contains v lines 4-10 of procedure *pathmatrix* assure that the length of that cycle is taken into account.

If v is not part of the simple cycle, then there exists a shortest path from v to the cycle. Be w the vertex on the cycle where this path ends. Be l_1 the length of the path and l_2 the length of the cycle. The minimal length of the path yields

$$l_1 \leq componentsize - l_2.$$

[9] See Errata, error no. 1, July 1997.

Therefore, vertex w appears in the periodicity matrix at least twice: In row l_1 and in row $l_1 + l_2$. Lines 11-21 of procedure *pathmatrix* take care that the length l_2 of the simple cycle is considered when the period is calculated.
The difference $newpathlength - pathlength$ in line 16 of procedure *pathmatrix* is not always a cycle length, but it is always a multiple of the period. In fact, be there two paths from v to w with lengths k_1 and k_2. We choose a path from w to v. Be m its length. Then there are a path from v to v with length $k_1 + m$ and a path from v to v with length $k_2 + m$. The difference of the two lengths, that is $k_2 - k_1$ must be divisible by the period. □

```
inworklist(v)

1    mark v;
2    for (each child w of v)
3        { if (unmarked w) inworklist(w);
4        };
5    insert v at the top of the work list;
```

Table 1: *Depth first search in forward direction to build a work list*

```
buildmemberlist(v)

1    for (each parent w of v)
2        { if (w not yet assigned to a component)
3              { mark w as assigned to actual component and insert into its member list;
4                unlink w from work list;
5                buildmemberlist(w);
6              };
7        };
```

Table 2: *Depth first search in backward direction to build a member list of vertices of a component*

```
mainprogram

1   for (each vertex v on the vertex list)
2       {if (v is not marked) inworklist(v);
3       };
4   while (top of work list is not empty)
5       {add new entry to component list;
6        v = top of the work list;
7        buildmemberlist(v);
8        if (member list empty);
9           {mark component list entry as no return and complete structure
10              with v as the only member of the component;
11          unlink v from work list;
12          }
13       else
14          {mark component list entry as return;
15          period = 0;
16          pathmatrix(1, v);
17          oldpathlength = 1;
18          while ((oldpathlength < componentsize) ∧ (period ≠ 1))
19              {for (each end vertex w of a path of length oldpathlength from v)
20                  {pathmatrix(oldpathlength + 1, w);
21                  };
22               oldpathlength = oldpathlength + 1;
23              };
24          allocate periodicity pointer array;
25          distribute vertices from member list to periodicity classes;
26          };
27      };
28  for (all components)
29      {for (all periodicity classes of each component)
30          {for (all vertices of each periodicity class)
31                  {distribute the entries of the child list to the class child list
32                      and the non-class child list;
33                   distribute the entries of the parent list to the class parent list
34                      and the non-class parent list;
35          };
36       };
```

Table 3: *Main program to construct the component/periodicity representation of a digraph*

```
pathmatrix(newpathlength, v)[a]

1    for (each child w of v)
2        { if ((v and w belong to the same component) ∧
                (the entry (newpathlength, w) in the path matrix is NULL))
3            { add the entry (newpathlength, w) to the path matrix;
4              if (w equals v)
5                  {
6                    if (period equals 0)
7                        {period = newpathlength;}
8                    else
9                        {period = gcd(period, newpathlength);};
10                 }
11             else
12                 { for (each pathlength < newpathlength)
13                     { if (the entry (pathlength, w) in the path matrix is not NULL)
14                         {
15                           if (period equals 0)
16                               { period = newpathlength − pathlength;}
17                           else
18                               { period = gcd(period, newpathlength − pathlength);};
19                         };
20                     };
21                 };
22           };
23       };
```

[a] See Errata, error no. 2, July 1997.

Table 4: *Construction of the path matrix and calculation of the period of a component*

5 Efficiency Considerations

The construction of the work list (lines 1-3 in table 3 and procedure *inworklist*, table 1) takes 1 processing operation for each vertex and 1 test operation for each arc of the digraph.

The construction of the memberlists of the components (lines 4-11 of table 3 and procedure *buildmemberlist*, table 2) again takes 1 processing operation per vertex and 1 test operation per arc.

All the above together costs $O(n^2)$ operations in the worst case, n being the number of vertices of the digraph.

Distributing the vertices of the member list is not explicitly described in table 3. It can be done searching for each vertex of the member list for the smallest path length in the path matrix. The complexity of this is $O(n^2)$ in the worst case. It can be reduced to $O(n)$ keeping a pointer to the first non null entry of each column of the path matrix.

The dominating efficiency factor are the construction of the path matrix and the calculation of the period (lines 18-22 of table 3 and procedure *pathmatrix*, table 4). In the worst case, the digraph consists of only 1 component and the number of children of each vertex is of order $O(n)$. So, to build up one row, $O(n)$ insertion operations and $O(n^2)$ tests have to be done. Each insertion to row k, takes $k-1$ periodicity tests. As there are $n-1$ rows to build this way the complexity in the worst case is $\sum_{k=2}^{n}(k-1)\cdot O(n)$ operations and $(n-1)\cdot O(n^2)$ tests yielding together $O(n^3)$. This is not satisfying and better periodicity algorithms should be searched for.

Remark: Stopping the path matrix construction as soon as aperiodicity is detected (line 18) does not reduce the algorithm's worst case complexity for aperiodic digraphs. The digraph in figure 9 is an example when A_1 is the root of the component.

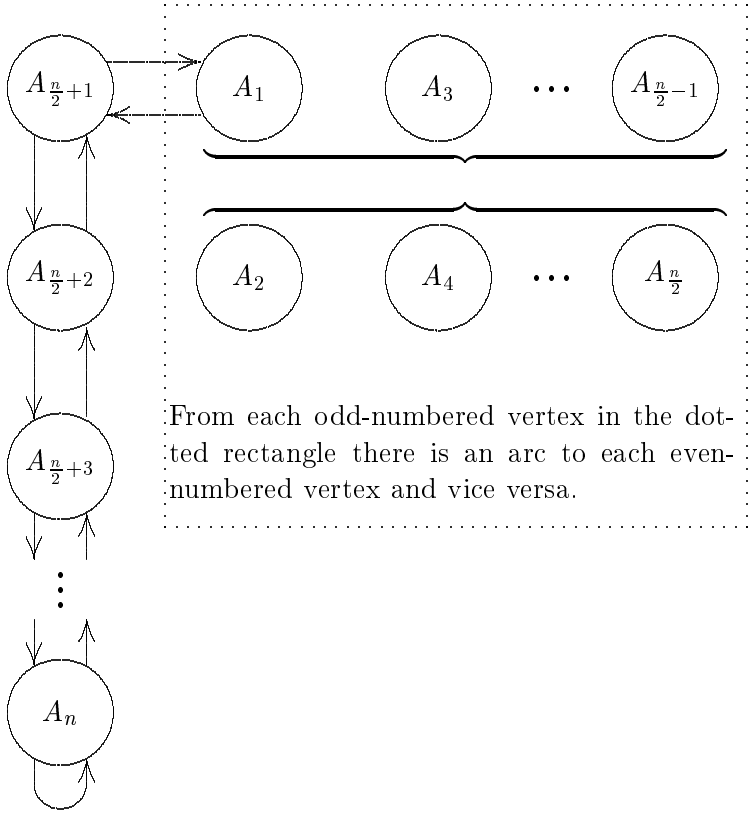

Figure 9: *An aperiodic digraph with $O(n^3)$ complexity to find the period*

6 Concluding Remarks

Passing from the vertices of a digraph to the strongly connected components yields a directed acyclic graph (dag), the *reduced digraph*. In the reduced digraph there is an arc from component $[A]$ to component $[B]$ ($[A] \neq [B]$) iff there exists an arc from a vertex in $[A]$ to a vertex in $[B]$.

If instead of building classes from mutually accessible vertices classes are built using periodicity according to theorem 3.1, a not so collapsed, intermediate digraph is obtained. We shall call it the *periodicity digraph* of the given digraph. Periodicity digraphs can be characterized as digraphs where at most 1 simple cycle passes through each vertex. They have a much simpler structure than general digraphs. This fact may be useful in problems like path finding.

Figure 10 shows an example of a digraph, figure 11 shows its reduced digraph, and figure 12 its periodicity digraph. In the figures, ovals mean proper components or periodicity classes respectively.

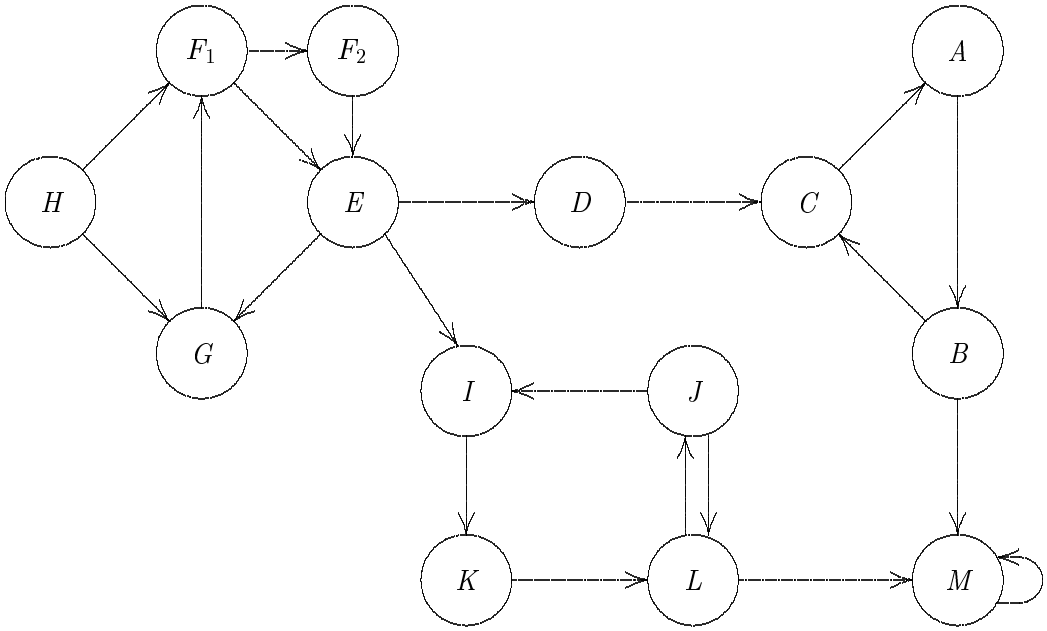

Figure 10: *General digraph*

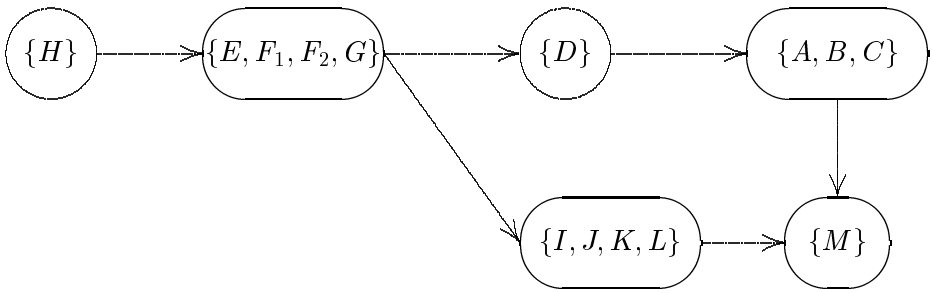

Figure 11: *Reduced digraph*

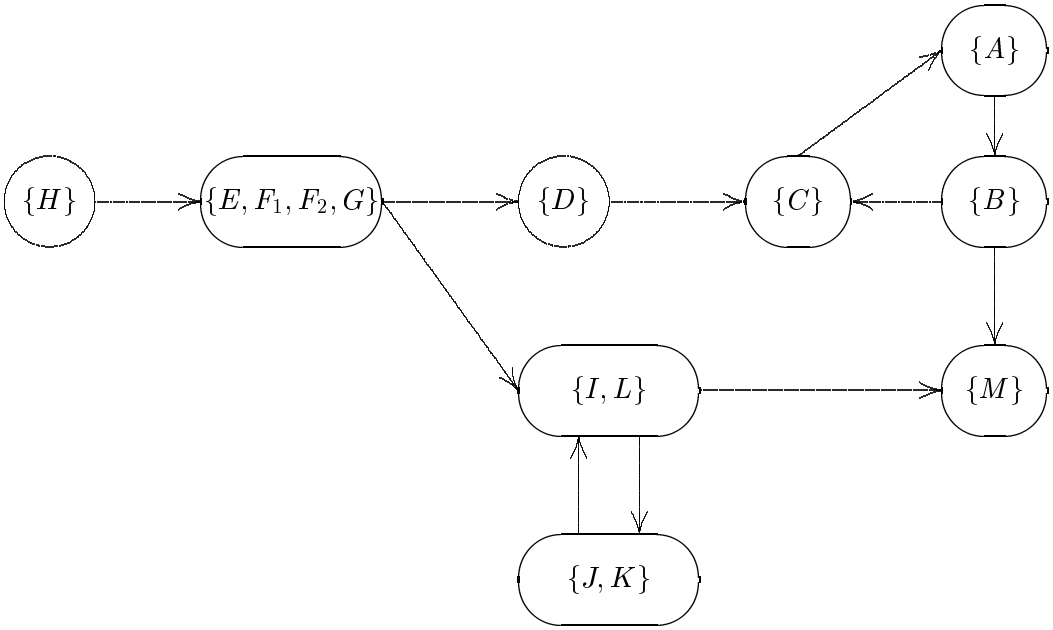

Figure 12: *Periodicity digraph*

References

Pages where cited are in parentheses.

[AhHU74] Aho, Alfred V. · Hopcroft, John E. · Ullman, Jeffrey D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Company, 1974. *(2)*

[AhHU83] Aho, Alfred V. · Hopcroft, John E. · Ullman, Jeffrey D. *Data Structures and Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Company, 1983. *(2, 10)*

[Ande70] Anderson, Sabra S. *Graph Theory and Finite Combinatorics.* Markham Mathematics Series. Markham Publishing Company, 1970. *(2)*

[Asmu87] Asmussen, Søren. *Applied Probability and Queues.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1987. *(2)*

[Boga83] Bogart, Kenneth P. *Introductory Combinatorics.* Pitman, 1983. *(2)*

[BoWe81] Böhm, Franz · Weise, Gotthart. *Graphen in der Datenverarbeitung.* Mathematik für Ingenieure. VEB Fachbuchverlag Leipzig, 1981. Anwendungen des Markierungsmodells in Technik und Ökonomie. *(2)*

[Carr79] Carré, Bernard. *Graphs and Networks.* Oxford Applied Mathematics and Computing Science Series. Clarendon Press · Oxford, 1979. *(2)*

[Chen90] Chen, Wai-Kai. *Theory of Nets: Flows in Networks.* John Wiley & Sons, 1990. *(2)*

[Chun67] Chung, Kai Lai. *Markov Chains With Stationary Transition Probabilities.* Die Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 2 edition, 1967. *(2)*

[Cinl75] Çinlar, Erhan. *Introduction to Stochastic Processes.* Prentice-Hall, Inc., 1975. *(2, 4)*

[FaKO81] Fahrmeir, Ludwig · Kaufmann, Heinz · Ost, Friedemann. *Stochastische Prozesse – Eine Einführung in Theorie und Anwendungen.* Mathematische Grundlagen. Carl Hanser Verlag, 1984. *(2)*

[Fell68] Feller, William. *An Introduction to Probability Theory and Its Applications – Volume I.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 3 edition, 1968. *(2)*

[Free83] Freedman, David. *Markov Chains.* Springer-Verlag, 1983. *(2)*

[Gibb85] Gibbons, Alan. *Algorithmic Graph Theory.* Cambridge University Press, 1985. *(2)*

[Gned87] Gnedenko, Boris W. *Lehrbuch der Wahrscheinlichkeitsrechnung.* Verlag Harri Deutsch, 1987. *(2)*

[GoMi84] Gondran, Michel · Minoux, Michel. *Graphs and Algorithms.* Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, 1984. *(2)*

[GrSt82] Grimmet, Geoffrey R. · Stirzaker, David R. *Probability and random processes.* Clarendon Press, 1982. *(2)*

[HoSa76] Horowitz, E · Sahni, S. *Fundamentals of Data Structures.* Computer Software Engineering Series. Pitman Publishing Limited, 1976. *(2)*

[IsMa76] Isaacson, Dean L. · Madsen, Richard W. *Markov Chains Theory and Applications.* Robert E. Krieger Publishing Company, Inc., 1976. *(2, 4, 6)*

[Jung90] Jungnickel, Dieter. *Graphen, Netzwerke und Algorithmen.* BI Wissenschaftsverlag, 2 edition, 1990. *(2)*

[KeSn76] Kemeny, John G. · Snell, J. Laurie. *Finite Markov Chains.* Undergraduate Texts in Mathematics. Springer-Verlag, 1976. *(2)*

[Koen90] König, Dénes. *Theory of Finite and Infinite Graphs.* Birkhäuser, 1990. Originally: Theorie der endlichen und unendlichen Graphen, Leipzig, 1936. *(2)*

[Kohl77] Kohlas, Jürg. *Stochastische Methoden des Operations Research.* Teubner Studienbücher Mathematik. B. G. Teubner, 1987. *(2)*

[Kohl87] Kohlas, Jürg. *Zuverlässigkeit und Verfügbarkeit.* Teubner Studienbücher Mathematik. B. G. Teubner, 1987. *(2)*

[LaJa79] Langrock, P. · Jahn, W. *Einführung in die Theorie der Markovschen Ketten und ihre Anwendungen.* Mathematisch-Naturwissenschaftliche Bibliothek. BSB B. G. Teubner Verlagsgesellschaft, 1979. *(2)*

[MaPf90] Mathar, Rudolf · Pfeifer, Dietmar. *Stochastik für Informatiker.* Leitfäden und Monographien der Informatik. B. G. Teubner, 1990. *(2)*

[MaRa91] Maurer, Stephen B · Ralston, Antony. *Discrete Algorithmic Mathematics.* Addison-Wesley Publishing Company, 1991. *(2)*

[Medh94] Medhi, Jyotiprasad. *Stochastic Processes.* John Wiley & Sons, 2 edition, 1994. *(2)*

[Mehl84] Mehlhorn, Kurt. *Data Structure and Algorithms 2: Graph Algorithms and NP-Completeness.* Monographs on Theoretical Computer Science. Springer-Verlag, 1984. *(2)*

[Nolt76] Noltemeier, Hartmut. *Graphentheorie mit Algorithmen und Anwendungen.* Walter de Gruyter, 1976. *(2)*

[ObWi76] Oberschelp, Walter · Wille, Detlef. *Mathematischer Einführungskurs für Informatiker.* Teubner Studienbücher Informatik. B. G. Teubner Stuttgart, 1976. *(2)*

[Osak92] Osaki, Shunji. *Applied Stochastic System Modeling.* Springer-Verlag, 1992. *(2)*

[Resn92] Resnick, Sidney I. *Adventures in Stochastic Processes.* Birkhäuser, 1992. *(2)*

[Revu84] Revuz, D. *Marcov Chains.* North-Holland, 1984. Revised Edition. *(2)*

[Rose74] Rosenblatt, M. *Random Processes.* Graduate Texts in Mathematics. Springer-Verlag, 1974. *(2)*

[Ross83] Ross, Sheldon. *Stochastic Processes.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1983. *(2, 4)*

[Sach70] Sachs, Horst. *Einführung in die Theorie der endlichen Graphen I.* Mathematisch-Naturwissenschaftliche Bibliothek. BSB B. G. Teubner, 1970. *(2)*

[Sedge91] Sedgewick, Robert. *Algorithmen.* Addison-Wesley Publishing Company, 1991. *(2)*

[Solb75] Solberg, J. J. A graph theoretic formula for the steady state distribution of finite Markov processes. *Management Science*, 21:1040 – 1048, 1975. *(2)*

[Stie95c] Stiege, G. *Angewandte Stochastische Prozesse.* Skripten des Fachbereichs Mathematik, Informatik, Naturwissenschaften. Universität Hildesheim, 1995. *(4, 6)*

Verzeichnis der Hildesheimer Informatik-Berichte

1/95 K. Nachtigall, St. Voget:
Optimierung von Periodischen Netzplänen mit Genetischen Algorithmen
(Januar 1995)

2/95 St. Voget:
Epistasis in Periodic Programs
(Januar 1995)

3/95 K.-J. Förster:
n Peano Constants in Gaussian Quadrature
(Januar 1995)

4/95 S. Ehrich, G. Mastroianni:
Stieltjes Polynomials and Lagrange Interpolation
(März 1995)

5/95 A. Lavrov:
Automated Simulation of Flexible Manufacturing Systems
(März 1995)

6/95 Jahresbericht Informatik 1994
(April 1995)

7/95 P. Köhler:
Error Estimates for Polynomial and Spline Interpolation by the Modulus of Continuity
(April 1995)

8/95 P. Köhler:
Asymptotically Best Spline Approximation by Quasi-Interpolation and Asymptotically Best Quadrature Formulae
(April 1995)

9/95 E. Best, M. Koutny:
A Refined View of the Box Algebra and Solving Recursive Net Equations
(April 1995)

10/95 E. Best, H. Fleischhack, W. Fraczak, R.P. Hopkins, H. Klaudel, E. Pelz:
A Class of Composable High Level Petri Nets and An M-Net Semantics of B(PN)2
(April 1995)

11/95 J. Desel (ed.):
Structures in Concurrency Theory (STRICT), Collection of Abstracts
(Mai 1995)

12/95 C. Eckert, H.-J. Klein, T. Polle:
7. Workshop 'Grundlagen von Datenbanken'
(Juni 1995)

13/95 T. Luba, H.-J. Bentz, J. Blieffert:
Information Retrieval in Sequence Databases Using Sparsely Coded Associative Memory
(Mai 1995)

14/95 E. Best, H. Fleischhack (eds.):
Programming Environment Based on Petri Nets
(Mai 1995)

15/95 B. Pfitzmann, M. Waidner:
Strong Loss Tolerance for Untraceable Electronic Coin Systems
(Juni 1995)

16/95 K. Diethelm:
Error Bounds for Compound Quadratures for Hadamard-type Finite-Part Integrals
(Juni 1995)

17/95 K. Diethelm:
Numerical Approximation of Finite-Part Integrals with Generalized Compound Quadrature Formulae
(Juni 1995)

18/95 S. Ehrich:
Practical Error Estimates for the Gauss-Kronrod Quadrature Formula
(Juli 1995)

19/95 S. Ehrich:
High Order Error Constants of Gauss-Kronrod Quadrature Formulas
(Juli 1995)

20/95 K. Nachtigall, St. Voget:
Optimization of the Integrated Fixed Interval Timetable
(Juli 1995)

21/95 St. Voget:
Speeding up a Genetic Algorithm for the Optimization of Periodic Networks
(Juli 1995)

22/95 J. Z. Zhou:
Densely Coded Matrix Model of Associative Memory
(Juli 1995)

23/95 K. Diethelm:
Definite Quadrature Formulae for Cauchy Prinicipal Value Integrals
(August 1995)

24/95 G. Stiege:
Periodicity: An Application of Digraphs to Markov Chains
(August 1995)