Optimal Parallel Algorithm for the Hamiltonian Cycle Problem on Dense Graphs

Elias Dahlhaus
Department of Computer Science, University of Bonn

Péter Hajnal †
Department of Computer Science, University of Chicago and
Bolyai Institute, University of Szeged, Hungary

Marek Karpinski ‡
Department of Computer Science, University of Bonn

† Reasearch supported by NSF grant NSF 5-2761
‡ Supported in part by Leibnitz Center for
research in Computer Science and the DFG grant KA 637/2-1.

Abstract

Dirac's classical theorem asserts that, if every vertex of a graph G on n vertices has degree at least $\frac{n}{2}$ then G has a Hamiltonian cycle. We give a fast parallel algorithm on a $CREW - PRAM$ to find a Hamiltonian cycle in such graphs. Our algorithm uses a *linear number of processors* and is optimal up to a polylogarithmic factor. The algorithm works in $\mathcal{O}(\log^4 n)$ parallel time and uses linear number of processors on a $CREW - PRAM$. Our method bears some resemblance to Anderson's RNC algorithm [An] for maximal paths: we, too, start from a system of disjoint paths and try to glue them together. We are, however, able to perform the base step (perfect matching) deterministically.

We also prove that a perfect matching in dense graphs can be found in NC^2. The cost of improved time is a quadratic number of processors.

On the negative side, we prove that finding an NC algorithm for perfect matching in slightly less dense graphs (minimum degree is at least $(\frac{1}{2} - \epsilon)|V|$) is as hard as the same problem for all graphs, and interestingly the problem of finding a Hamiltonian cycle becomes NP-complete.

1. Introduction

In recent years parallel computation has attracted a great deal of attention in the theory of algorithms. Some problems, easy to solve sequentially in polynomial time, became non-trivial questions in parallel computation. The maximal independent set problem was one of them. The problem is now known to be in NC ([KW],[Lu],[ABI],[GS]).

Other important problems like maximum matching (matching of maximum size) are still not known to be in NC. There exist methods ([Lo1], [KUW],[MVV]) which show that matching is in RNC (random parallel polylog time) (in fact, in Las Vegas-NC [Ka]). One relaxation of the maximum matching problem is maximal matching (maximal with respect to inclusion). This is obviously in NC, as a consequence of the result for maximal independent sets. A more efficient algorithm can be found in [IS].

In the case of maximal independent set we had an NP-complete problem (maximum independent set) and we relaxed it. This way the problem became solvable in NC. There are other questions that arise in this way. The Hamiltonian path problem can be relaxed to the maximal path problem.

R. Anderson [An] presented a reduction of the maximal path problem to matching. The reduction implies that the maximal path problem is in RNC. His algorithm starts out from a system of paths and glues them together. This way one can reduce the problem to smaller ones. He and A. Aggarwal [AA] extended this method to the problem of finding a depth first search tree in NC.

Another simplification of a problem might be a restriction on the possible inputs. Depending on the problem, different classes of input have been studied. For example for maximum matching there are several NC algorithm which work for special classes ([DK1],[GK]). We call graphs $G = (V, E)$ with minimal degree at least $\frac{|V|}{2}$ *dense*. Recently, the complexity of problem for dense graphs has received growing attention ([Br],[Ed]).

The Hamiltonian circuit problem is NP-complete. There are known classes of graphs where all the members are Hamiltonian. One class is the tournaments (complete oriented graphs) (see [So]). Another class was found by Dirac ([Di],[Be],[Bo],[Lo2]) who showed that dense graphs have Hamiltonian cycles. At FOCS'87 M. Goldberg proposed the problem: Is there any NC algorithm which finds a Hamiltonian cycle for dense graphs? In this paper we answer Goldberg's question affirmatively, giving an optimal up to polylogarithmic factor algorithm. The earlier papers [DK3] and [H] gave a non-optimal NC solution for this problem.

Although a Hamiltonian cycle induces a perfect matching (for even n), we shall present a separate algorithm for the perfect matching problem. The reason is that maximum matching is a fundamental problem ([LP]) and the algorithm for it is simpler and is faster (NC^2), while the Hamiltonian cycle algorithm is in NC^3. Although both problems, Hamiltonian cycle, and perfect matching can be computed in $O(\log^4 n)$ parallel time, and linear number of processors, the perfect matching algorithm enjoys much better constant factors, and deserves an independent analysis. On the other hand we present an NC reduction of

the perfect matching problem in a slightly larger class to the general case, showing that our choice of input class is essentially at the border of what seems feasible in NC to date. In the literature there are more known reductions this type ([KL],[DK2]).

Section 2 introduces basic notation and terminology. In Section 3 we discuss the parallel complexity of maximal matching and describe our NC^2 algorithm for the perfect matching problem in dense graphs. In Section 4 we present a parallel algorithm for the construction of a Hamiltonian cycle in dense graphs.

2. Definitions and notation

We use standard graph theoretical notation and terminology. We refer the reader to [Lo2]. $G = (V, E)$ will always denote the input of the algorithm, i.e., a simple graph, with vertex set V, edge set E, and minimum degree at least $\frac{n}{2}$, where $n = |V|$ is the number of nodes. $d(u)$ is the degree of the node u. For $S \subset V$ we use $d_S(u)$ to denote the nuber of edges joining the from node u to S. A *matching* of (V, E) is a subset M of disjoint edges. A matching is *maximal* if it is not a proper subset of any matching. It is *maximum* if it has maximum cardinality among all matching of G.

3. Finding a perfect matching in dense graphs

We need as a subroutine, the following result of Israeli and Shiloach [IS]. We give a brief account of their algorithm.

Procedure **Approximate_matching**
Given: G, a simple graph.
Compute: A vertex cover A of G and a matching M lying in A covering at least half of the nodes of A.

After this it is easy to find a maximal matching of the given graph by iterating the procedure above. The analysis is summarized in the following theorem.

Theorem 3.1. *([IS]) (i) Procedure Approximate_matching can be implemented in $\mathcal{O}(\log^3 n)$ parallel time with $\mathcal{O}(m+n)$ processors on a $CREW - PRAM$.*
(ii) Program Maximal_matching can be implemented in $\mathcal{O}(\log^4 n)$ parallel time with $\mathcal{O}(m+n)$ processors on a $CREW - PRAM$. ∎

Now we present an NC^2-algorithm constructing a perfect matching in dense graphs.

Find_perfect_matching
Given: A dense graph $G = (V, E)$, where $|V|$ is even.
Compute: A perfect matching of G.

1) Compute a maximal matching M_1 of G.
{Each edge contains at least one vertex appearing in M_1. At least $\frac{|V|}{2}$ vertices belong to an edge of M_1. }

2) Let $\{x_1, \cdots, x_{2k}\}$ be the set of vertices of G not belonging to an edge of M_1 and define $G' = (V', E')$ as follows:
The vertex set V' consists of the edges of M_1 and of the unordered pairs $x_{2i-1}x_{2i}$, $i = 1, \ldots, k$. The edge set is defined as follows: $x_{2i-1}x_{2i}$ and $yz \in M_1$ are joined by an edge in E' iff $x_{2i-1}y$ and $x_{2i}z \in E$ or $x_{2i-1}z$ and $x_{2i}y \in E$.
{Note that G' is bipartite.}
3) Compute a maximal matching M_2 of G'.
{Each vertex of G' of the form $x_{2i-1}x_{2i}$ belongs to an edge of M_2.}
4) For each i, $1 \leq i \leq k$, consider $uv \in M_1$, adjacent in M_2 to $x_{2i-1}x_{2i}$. W.l.o.g. $x_{2i-1}u, x_{2i}v \in E$. Delete uv from M_1 and add $x_{2i-1}u$ and $x_{2i}v$ to M_1.
{M_1 is transformed into a perfect matching.}
5) Output M_1.

End Find_perfect_matching

Theorem 3.2. *(i) For each dense graph of an even number of vertices a perfect matching can be constructed in NC^2*
(ii) For each dense graph of an even number of vertices a perfect matching can be constructed in $\mathcal{O}(\log^4 n)$ time on a $CREW - PRAM$ using linear number of processors.

Proof. In the algorithm Find_perfect_matching, the most expensive step is finding maximal matching. If we use Luby's algorithm for the line graph in order to accomplish this we obtain an NC^2 implementation of our algorithm. If we use the maximal matching algorithm of Israeli and Shiloach then we use up $\mathcal{O}(\log^4 n)$ time but save on processors, yielding (ii). ∎

The next question we consider is the parallel complexity of matching for graphs of a minimal degree $\alpha|G|$, where $\alpha < \frac{1}{2}$.

Theorem 3.3. *For $\alpha < \frac{1}{2}$ the existence problem for a perfect matching restricted to graphs $G = (V, E)$, s.t. the minimal degree is at least $\alpha|V|$, is NC-hard for the general matching problem. This means that an NC-algorithm for the matching problem restricted to graphs of minimal degree $\alpha|V|$ would imply an algorithm for the general perfect matching problem.*

Proof. Let $G = (V, E)$ be any graph. We construct a graph $G' = (X \dot{\cup} Y \dot{\cup} V, E')$ as follows: X and Y are sets of equal size, Y is an independent set in G' and G is an induced subgraph of G'. Every node in X is adjacent to all nodes in G'. We choose the size of X to be $|X| = \lceil \frac{\alpha}{1-2\alpha}|V| \rceil$. ∎

A similar proof technique was used by A. Broder [Br], when he showed that determining the permanent on "dense" bipartite graphs is $\#P$-hard.

4. Finding a Hamiltonian circuit in dense graphs

4.1. The outline of the algorithm

First, our algorithm finds a Hamiltonian path in the given graph. Having done so it will be easy to finish the algorithm, i.e., to find a Hamiltonian cycle.

The main idea of finding a Hamiltonian path is that we maintain a path system that covers the graph, i.e., a set of disjoint paths covering the entire vertex set. The algorithm consists of phases. In each phase we try to merge different paths. This way we can reduce the number of paths by a constant factor. In order to merge paths we use a special operation (to be defined later).

We want to merge several paths into others in parallel. We might have conflicts during the parallel merging. To overcome this problem for each path we want to find several ways to merge into another path.

$\{P_i\}_{i=0}^{k}$ will denote a set of paths in G such that $\cup_i V(P_i) = V(G)$ and the $V(P_i)$ are disjoint. We will refer to this as a *path-cover* of G.

4.2. Sociable paths

In this section we give the formal definition of our elementary merging operation and introduce different types of paths. The basic idea of these types comes from Lemma 4.2.5, which says that if the endpoints of a path are connected with a lot of edges to another path then there are several possible ways to merge that path into the other.

Definition 4.2.1. Let $P = (u_1, \ldots, u_l)$ and $Q = (v_1, \ldots, v_m)$ be two disjoint paths If u_1v_i and u_lv_{i+1} are edges of the graph then $v_1 \ldots v_i u_1 \ldots u_l v_{i+1} \ldots v_m$ is a path. If u_1v_{i+1} and u_lv_i are edges of the graph then $v_1 \ldots v_i u_l \ldots u_1 v_{i+1} \ldots v_m$ is a path. In either case we say that we *merged P into Q along the edge v_iv_{i+1}*. We call this step an *elementary merging operation.*

We need some other operations too.

Definition 4.2.2. Let P and Q be two disjoint path. If there is an edge connecting an endpoint of P and an endpoint of Q, then we can use this edge to concatenate the two paths. We call this operation a *concatenation.*

Definition 4.2.3. Let C and D be two disjoint cycles. If there is an edge connecting a node from C and a node from D then we can use this edge to get a paths which passes through all the vertices of C and D. We call this operation a *cycle merging.*

Definition 4.2.4. Let P be a path in G. Let u, v be the two endpoints of P. We call P *sociable* if

$$d_{V(P)}(u) + d_{V(P)}(v) + 1 \leq |V(P)|.$$

We say a path P is *introverted* if it is not sociable.

We need the following lemma.

Lemma 4.2.5. *Let P and Q be disjoint. Let u, v be the endpoints of P. Let us assume that there are no edges from any endpoint of P to an endpoint of Q. Then Q has at least*

$$d_{V(Q)}(u) + d_{V(Q)}(v) + 1 - |V(Q)|$$

edges along which one can merge P into Q via an elementary merging operation.

Proof. Let $Q = (q_1, q_2, ..., q_l)$. Let $d = d_{V(Q)}(u)$ and $e = d_{V(Q)}(v)$. Let $\{ q_{i_1}, ..., q_{i_d} \}$ be the neighborhood of u in $V(Q)$. By assumption, $1 < i_1$ and $i_d < l$. Let $F = \{ q_{i_1-1}, q_{i_1+1}, q_{i_2+1}, ..., q_{i_d+1} \}$. F contains $d+1$ different nodes from Q. If v is adjacent to one of them then one can easily find an edge where elementary merging can be performed. Actually we can assign different edges of Q to different elements of F in such a way that an edge between v and an element of F gives a possible elementary merging operation along the corresponding edge. If $d_{V(Q)}(u) > |V(Q) - F|$ then one can find a way to merge. Actually there will be at least $e - (l - (d+1))$ edges joining v to F. This proves the lemma. ∎

The lemma above has the following important consequence for the path covering.

Corollary 4.2.6. *Let $\{ P_i \}_{i=0}^{k-1}$ be a path-cover of G. We assume that P_0 is sociable and there are no edges going from its endpoint to the endpoints of other covering paths. Then there are at least k edges on $\cup_{i=1}^{k-1} P_i$ along which one can merge P_0 into another path.*

Proof. Let u, v be the endpoints of P_0. Let $d_i (i = 0, ..., k-1)$ be the degree of u toward P_i and let $e_i (i = 0, ..., k-1)$ be the corresponding degrees of v. Let n_i be the number of nodes on P_i. Using this notation we have

$$\sum_{i=0}^{k-1} d_i + \sum_{i=0}^{k-1} e_i = d(u) + d(v) \geq \frac{n}{2} + \frac{n}{2} = n = \sum_{i=0}^{k-1} n_i.$$

After rearrangment we get

$$\sum_{i=0}^{k-1} (d_i + e_i + 1 - n_i) \geq k.$$

We can delete the nonpositive terms in the sum and the inequality remains valid. We assumed that P_0 is social so during the simplification above the term $d_0 + e_0 + 1 - n$ is at most 0 and will be deleted. After the deletion we have

$$\sum_{i=1}^{k-1} max\{ d_i + e_i + 1 - n_i, 0 \} \geq k.$$

By Lemma 4.2.5 we get the result. ∎

In the case of introverted paths we need an additional trick. We need the following generalization of Corollary 4.2.6, a direct consequence of the proof above.

Lemma 4.2.7. *Let S be a subset of $V(G)$ and $\{ P_i \}_{i=1}^{k-1}$ be a path-cover of $V(G) - S$. Let P be a path in S with endpoints u, v. Let us assume that $d_S(u) + d_S(v) + 1 - |S| \leq 0$ and that there are no edges $\{ u, v \}$ to any endpoint of the path-cover. Then there are at least k edges on $\cup_{i=1}^{k-1} P_i$ along which P can be merged into a covering path.* ∎

4.3. Introverted paths

First we prove that an introverted path can be closed to a cycle, i.e., the graph induced on the vertex set of an introverted path has a Hamiltonian cycle.

Lemma 4.3.1. *Let P be an introverted path between endpoints u and v. Then one of the following three statements is true:*
(a) P is a single node or edge.
(b) There exists an edge between the two endpoints of P.
(c) There exists a neighbor u' of u on P and a neighbor v' of v on P such that u' and v' are adjecent via an edge of P and u' is between v' and v on P.

Proof. Let us assume that P has more than 2 nodes. Let $\{ w_1, ..., w_l \}$ be the vertex set of P ($w_1 = u$ and $w_l = v$). The order on the path is the same as the order of the indices. Let $d = d_{V(P)}(u)$ and $e = d_{V(P)}(v)$. We can assume that there is no edge between u and v. Let $\{ w_2, w_{i_2}, ..., w_{i_d} \}$ be the neigborhood of u. If (c) is not valid then $\{ w_1, w_{i_2 - 1}, ..., w_{i_d - 1}, w_l \}$ cannot be adjecent to v. This means that $l - (d + 1) \geq e$. So P is social. This contradicts our assumption. ∎

In Case (a), the path has only endvertices. In the other two cases one can easily find a Hamiltonian cycle in the graph induced by the introverted path. This allows cycle merging.

4.4. The general case

We need some additional tricks to handle the general case.

The algorithm starts with an initial path-covering. We shall require each path to have at least 2 nodes. Because of this, the initialization step is not totally obvious. After initialization, using the results of the previous sections, we reduce the number of paths by a constant factor.

Procedure **Initialization**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G.
Compute: A path-cover of G such that each path in the cover has at least two points.

1) In the case of odd vertex set add a new node to the graph and connect it to all old vertices.
2) Find in parallel a perfect matching in the extended graph.
3) In the case of even vertex set output this matching as a path cover. If the vertex set is odd then the perfect matching found gives us a partial matching of the original graph. Adding one edge we extend this to a path cover.

End Initialization

Procedure **Reduce_path_cover**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G, and a path-cover of G with at least 2 paths.
Compute: A new path-cover with at most $\frac{15}{16}$ as many paths.

1) In parallel classify each path as sociable or introverted.
2) Arbitrarily divide the set of introverted paths into pairs of *mates* (with at most one path left alone). Call two mate ***linked*** if there is an edge between them.
3) Repeat until there are only unlinked pairs of introverted paths.
   For all linked pairs of introverted path do a)-d):
   a) For each pair find an edge between a path and its mate.
   b) Find a Hamiltonian cycle on the vertex set of each introverted path having at least 3 nodes.
   c) Do cycle merging.
   d) Devide the set of new introverted paths into pairs of mates.
   { When we exit this loop we have a path-cover of G. The cover consists of sociable paths and pairs of independent introverted paths. We will refer to the sociable paths and to the pairs of introverted paths as *generalized components* of the cover.}
4) Consider the subgraph induced by the endpoints of the covering paths. Find an approximate matching in this subgraph. Using these edges concatenate paths. Kill all resulting cycles by deleting one new edge of each.

{ The approximate matching procedure (see in Section 3) in addition gives us a vertex cover. We refer to a generalized component that lies outside this vertex cover as an *untouched component.* At this point there are no edges between endpoints of different paths belonging to untouched components. In the rest of the algorithm, we shall attempt to merge untouched paths into other paths. }

5) For every untouched component do the following: If it is a pair of introverted paths then take the smaller one, otherwise take the component itself (a sociable path). Now we have a path and we want to merge it into another one. Find all the edges on the path system along which an elementary merging can be performed.
6) For each untouched component we have many possible merging operations. For different untouched components find a merging from Step 5 which uses different edges. We will see that this can be solved in the following way. We construct an auxiliary bipartite graph H_1 between the untouched components and the edges along the paths. A component will be connected to an edge iff along it there is a possible merging (found in Step 5) into the corresponding path. An approximate matching of H_1 will give the 1-1 map from a constant fraction of untouched components into edges.
7) Perform as many elementary operations, found above, as possible. This can be done as follows. Make the following auxiliary directed graph H_2. The nodes will correspond to paths in the cover. There is an edge going from a path P to a path Q iff one of the merging operations, found in Step 6 merges P into Q. In this digraph every node has outdegree 1 or 0. Each component of such a digraph contains at most one cycle. We kill one edge of each cycle and perform the merging operations corresponding to the remaining edges in parallel.

End Reduce_path_cover

Program **Find_Hamiltonian_circuit**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G.
Compute: A Hamiltonian circuit of G.

1) Initialization
2) Repeat while there are at least two paths
   a) Reduce_path_cover
   {At this point of the algorithm we have a Hamiltonian path $\{v_1, ..., v_n\}$ of G.}
3) If v_1v_n is not an edge of G then find a pair of edges of the form v_1v_{i+1} and v_nv_i.
   {We will see that in this specific case this kind of edge pair exists in the graph.}
4) Output the cycle $v_1...v_nv_1$ or $v_{i-1}...v_1v_{i+1}...v_nv_iv_{i-1}$ according to the case in Step 3.

End Find_Hamiltonian_circuit

At several points of the proof of correctness we will need the following very simple but useful remark.

Lemma 4.4.1. *Let H be a bipartite graph between sets A and B. Let us assume that every node in A has degree at least $|A|$. Then every maximal matching of H covers the whole set A.* ∎

The following lemma shows how the number of paths in the path-cover changes during the procedure Reduce_path_cover.

Lemma 4.4.2. *Let us run the procedure Reduce_path_cover once. Let p be the number of paths at the beginning. Let p' be the number of paths and c be the number of generalized components after executing Step 3. Let t be the number of components touched in Step 4. Then*
(i) In Step 4 after $\log p = \mathcal{O}(\log n)$ iterations we stop with $c \geq \frac{p'}{2}$ generalized components.
(ii) The procedure outputs a path-cover with at most $\frac{15}{16}p$ paths. ∎

The previous lemma implies the correctness of the algorithm.

Theorem 4.4.3. *The program Find_Hamiltonian_cycle terminates in $\mathcal{O}(\log^4 n)$ time on a $CREW - PRAM$, uses linear number of processors and, outputs a Hamiltonian cycle of G.* ∎

4.5 Lower densities

Now we consider the class of α-dense graphs $G = (V, E)$ for $\alpha < \frac{1}{2}$, i.e. graphs with minimal degree of at least $\alpha|V|$.

Theorem 4.5.1. *For every $\alpha < \frac{1}{2}$, the Hamiltonian cycle problem restricted to α-dense graphs is NP-complete.*

Proof. We reduce the existence problem of a Hamiltonian path in a graph G to the existence problem of a Hamiltonian cycle in an α-dense graph G'. We construct $G' = (V', E')$ from a given $G + (V, E)$ as follows:

- the vertex set of G' is a disjoint union of the Vertex set of G and two other sets, $V' = V \dot{\cup} X \dot{\cup} Y$, where $|X| = |Y| + 1 = k = \lceil \frac{\alpha}{1-2\alpha}(|V| + 1) \rceil$.
- the edge set of G' consists of: (i) the edge set of G, (ii) thge complete bipartite graph between V and X, (iii) the complete bipartite graph between X and Y

Each Hamiltonian cycle in G' must have a subpath $x_1 y_1 x_2 y_2 \ldots x_{k-1} y_{k-1} x_k$, where $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_{k-1}\}$. The rest of the Hamiltonian path passes through G. Therefore G has a Hamiltonian path if and only if G' has a Hamiltonian cycle. Easy to check that G' is α-dense by setting the size of X and Y as we did. ∎

5. Conclusion and open problems

The sequential deterministic algorithm computing a Hamiltonian cycle for any dense graph seems related to the probabilistic solution of Angluin and Valiant [AV] which computes a Hamiltonian cycle with high probability for any graph if it has one. One would hope to be able to turn Frieze's [Fr] probabilistic parallel algorithm into a deterministic algorithm. But we were not successful in dividing any dense graph into two dense graphs of nearly equal size by an NC-algorithm.

There are more sufficient conditions for having Hamiltonian cycles ([Be],[Bo],[Lo2]). It would be desirable to extend our results to the corresponding classes of graps.

Another relaxation similar to the Hamiltonian path problem is a maximal cycle problem. One can fix some elementary operations for enlarging a cycle. One example is the following. Let us assume that some node not on the cycle is adjacent to two neighbors on the cycle. One can then easily merge this node into the cycle. Is there any NC algorithm which finds a maximal cycle in a graph, maximal respect to this operation?

Acknowledgement

Dominic Welsh originally drew our attention to the fast parallel computation problems on dense graphs by making us acquainted with Edward's results [Ed]. We are grateful to Mark Goldberg and Christos Papadimitriou for proposing the problem of constructing a Hamiltonian cycle for dense graphs in parallel. We would also like to thank Ephraim Korach for some hints on the literature, and László Babai, Howard Karloff, Richard Karp, János Simon, Eli Upfal, and Avi Wigderson for many interesting conversations.

References

[An] R.J. Anderson, A parallel algorithm for the maximal path problem, *Combinatorica*, 7 (1987), 315-326.

[AA] A. Aggarwal and R.J. Anderson, A random NC-algorithm for depth first search, *Proc. 19th ACM STOC*, 1987, pp. 325-334.

[ABI] N. Alon, L. Babai and A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, *Journal of Algorithms*, 7 (1986), pp. 567-583.

[AV] D.Angluin and L. Valiant, Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings,*J. Computer Syst.*, 18 (1979), 155-193.

[AIS] B.Awerbuch, A. Israeli and Y. Shiloach, Finding Euler circuits in logarithmic parallel time, *Proc. 16th ACM STOC*, 1984, pp. 249-257.

[Be] C. Berge, *Graphs and Hypergraphs*, North-Holland - American Elsevier, 1973.

[Bo] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.

[Br] A. Z. Broder, How hard Is it to Marry at Random, *Proc. 18th ACM STOC*, 1986, pp. 50-58.

[Co] S. Cook, A Taxonomy of Problems with Fast Parallel Algorithms, *Information and Control*, 64 (1985), pp. 2-22.

[DK1] E. Dahlhaus and M. Karpinski, The Matching Problem for Strongly Chordal Graphs Is in NC, *Research Report No. 855-CS*, University of Bonn 1986.

[DK2] E. Dahlhaus and M. Karpinski, Perfect Matching for Regular Graphs is AC^0-Hard for the General Matching Problem,*Research Report No. 858-CS*, University of Bonn 1986.

[DK3] E. Dahlhaus and M. Karpinski, Parallel construction of perfect matchings and Hamiltonian cycles on dense graphs, Research Report No. 8518-CS, University of Bonn 1987.

[Di] G.A. Dirac, Some theorems on abstract graphs, *Proc. London Math. Soc.*, 2 (1952), 69-81.

[Ed] K. Edwards, The Complexity of Coloring Problems on Dense Graphs, *Theoretical Computer Science*, 43 (1986), pp. 337-343.

[Fr] A. M. Frieze, Parallel Algorithms for Finding Hamiltonian Cycles in Random Graphs, *Information Processing Letters*, 25 (1987), pp. 111-117

[GJ] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman & Company, San Francisco 1979.

[H] P. Hajnal, Fast parallel algorithm for finding a Hamiltonian cycle in dense graphs, *Technical Report 88-003-CS*, Univ. of Chicago, 1988.

[HM] D. Hembold and E. Mayr, Two-Processor Scheduling Is in NC, in: VLSI-Algorithms and Architectures, Makedon et. al. ed., pp. 12-25.

[GK] D. Yu. Grigoriev and M. Karpinski, The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanents Is in NC, *Proc. 28th IEEE FOCS*, 1987, pp. 166-172.

[Go] L. Goldschlager, Synchronous Parallel Computation, *Journal of the ACM*, **29** (1982), pp. 1073-1086.

[GS] M. Goldberg and T. Spencer, A new parallel algorithm for the maximal independent set problem, *Proc. 28th IEEE FOCS*, 1987, pp. 161-165.

[IS] A. Israeli and Y. Shiloach, An improved parallel algorithm for maximal matching, *Inf. Proc. Letters*, **22** (1986), 57-60.

[Ka] H. Karloff, A Las Vegas RNC algorithm for maximum matching, *Combinatorica*, **6** (1986), 387-392.

[KUW] R.M. Karp, E. Upfal and A. Wigderson, Constructing a perfect matching is in random NC, *Combinatorica*, **6** (1986), 35-48.

[KW] R.M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *JACM*, **32** (4) (1985), 762-773.

[KL] M. Karpinski and A. Lingas, Subtree Isomorphisms and Bipartite Matchings Are Mutually NC-Reducible, *Research Report No. 856-CS*, University of Bonn, 1986.

[LPV] G. Lev, M. Pippenger and L. Valiant, A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE-Transactions on Computers*, C-30 (1981), pp. 93-100.

[Lo1] L. Lovász, Determinants, matchings, and random algorithms, *Foundamentals of Computation Theory, FCT '79, (ed. L. Budach), Akademie-Verlag Berlin* 1979, 565-574.

[Lo2] L. Lovász, *Combinatorial Problems and Exercises*, North-Holland 1979.

[LP] L. Lovász and M. Plummer, *Matching Theory*, Mathematical Studies, Annals of Discrete Mathematics Vol. 25, North Holland, Amsterdam 1986.

[Lu] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. COMPUT.*, **15** (1986), 1036-1053.

[MVV] K. Mulmuley, U. Vazirani and V. Vazirani, Matching Is as Easy as Matrix Inversion, Combinatorica, **7** (1) (1987), 105-131.

[PU1] D. Peleg and E. Upfal, The Token Distribution Problem, *Proc. 27th IEEE FOCS*, 1986, pp. 418-427.

[PU2] D. Peleg and E. Upfal, Constructing Disjoint Paths on Expander Graphs, *Proc. 19th ACM STOC*, 1987, pp. 264-273.

[So] D. Soroker, Fast Parallel Algorithms for Finding Hamiltonian Paths and Cycles in a Tournament, *Report No. UCB/CSD 87/309*, University of California, Berkeley 1986.

[SS] E. Shamir and A. Schuster, Parallel Routing in Networks: Back to Circuit Switching?, Manuscript, 1986.