Title
Fast optimal parallel algorithms for maximal matching in sparse graphs

Authors
Asuri, Hari S.
Dillencourt, Michael B.
Eppstein, David
et al.

Publication Date
1992

Peer reviewed

Fast Optimal Parallel Algorithms for Maximal Matching in Sparse Graphs

Hari S. Asuri[*†]
Michael B. Dillencourt[*‡]
David Eppstein[*]
George S. Lueker[*†]
Mariko Molodowitch[**]

Technical Report 92-01

January, 1992

ABSTRACT

We present optimal parallel algorithms to find maximal matchings in two classes of sparse graphs, namely, graphs with bounded degree and those which, for some p, are forbidden to have the clique K_p as a minor. This latter class is quite large; for example, planar graphs satisfy this condition, as do graphs with any fixed bound on their genus. Our algorithms use $O(\log n)$ time on a CRCW PRAM for both classes of graphs. On an EREW PRAM, our algorithms use $O(\log n)$ time for graphs with bounded degree and $O(\log n \log^* n)$ time for graphs with forbidden K_p-minors.

* Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92717.

† The support of the National Science Foundation under Grant CCR 89-12063 is gratefully acknowledged.

‡ The support of a UCI Faculty Research Grant is gratefully acknowledged.

** Computer Science Department, California State University at Fullerton, Fullerton, CA 92634.

Fast Optimal Parallel Algorithms for Maximal Matching in Sparse Graphs

Hari S. Asuri[†]
Michael B. Dillencourt[‡]
David Eppstein
George S. Lueker[†]
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Mariko Molodowitch
Computer Science Department
California State University at Fullerton
Fullerton, CA 92634

1. Introduction

Graph matching is an important and well-studied problem (see [Edmo65], [HK73], [Gali86]). A *matching* for a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that no two edges in M share a vertex. The problem of finding a maximum cardinality matching has been widely studied (see [Gali86] for a survey). It is known that this problem is in RNC ([KUW86], [MVV87]), but whether it is in NC or not is open.

In this paper, we consider an easier problem called the maximal matching problem. A *maximal matching* is a matching M such that no other matching M' properly contains M. Thus if M is maximal, each unmatched edge must share one of its vertices with a matched edge. Finding a maximal matching has applications in approximation algorithms for vertex cover [GJ79] and also serves as a good first step for finding a maximum matching [Gali86].

There is a simple and well-known sequential linear-time algorithm for maximal matching. We start with a set of unmarked vertices and an empty match set. We then repeatedly choose any edge which has neither endpoint[1] marked, add it to the match set, and mark its two endpoints. We stop when no such edge remains. Finding a maximal matching quickly in parallel is a considerably more difficult problem. Israeli and Shiloach [IS86] presented an $O(\log^3 n)$ CRCW PRAM algorithm using $O(m+n)$ processors to find a maximal matching in general graphs with n vertices and m edges. For planar graphs, the running time of the Israeli and Shiloach algorithm can be reduced to $O(\log^2 n)$ [GPS87]. Goldberg, Plotkin, and Shannon [GPS87] presented an $O(\log n \log^* n)$ CRCW PRAM algorithm and an $O(\log^2 n)$ EREW algorithm for maximal matching in planar graphs, each using $O(n)$ processors.

---

† The support of the National Science Foundation under Grant CCR 89-12063 is gratefully acknowledged.

‡ The support of a UCI Faculty Research Grant is gratefully acknowledged.

1 By the *endpoints* of an edge (v, w) we mean the two vertices v and w.

The algorithms presented here are improvements over the [GPS87] algorithm for planar graphs; in addition, these algorithms cover a much larger class of graphs. We consider two classes of sparse graphs: (1) bounded-degree graphs and (2) graphs which, for some fixed p, do not contain K_p as a minor. The latter class of graphs is a substantial generalization of the class of planar graphs; in particular, it contains all bounded-genus graphs. Note that neither one of these classes ((1) and (2)) contains the other. For example, no planar graph contains K_5 as a minor, while there are planar graphs with vertices of unbounded degree. On the other hand, there exist bounded degree graphs with K_p-minors for arbitrary p. Examples of such graphs, for any p, can be constructed by starting from K_p and then by expanding the vertices suitably such that the vertices of the resulting graph are of degree at most 3. For these two classes of graphs, using a CRCW PRAM, we find a maximal matching in $O(\log n)$ time with $O(n/\log n)$ processors. Using an EREW PRAM, we find a maximal matching for graphs of bounded degree in $O(\log n)$ time using $O(n/\log n)$ processors, and for graphs with forbidden K_p-minors in $O(\log n \log^* n)$ time using $O(n/\log n \log^* n)$ processors.

2. Preliminaries

Let $G = (V, E)$ be an undirected graph. We let $|V| = n$ and $|E| = m$. K_n refers to a complete graph on n vertices. An *elementary contraction* of G is obtained by selecting two adjacent vertices $v, u \in V$ and merging them; more formally, we replace v, u, and all incident edges by a new vertex w which is adjacent to all vertices which were adjacent to v or u. G' is said to be a *contraction* of G if G' is obtained from G through a series of elementary contractions. A graph H is said to be a *minor* of G if H is a contraction of a subgraph of G. A graph G is *K_p-minor-free* if K_p is not a minor of G.

Our algorithms make use of several other well-known parallel algorithms. In the *list ranking* problem we are given a list and required to determine the distance of each of the items from the end of the list. List ranking can be performed optimally in $O(\log n)$ time deterministically on an EREW PRAM (see [AM88] and [CV88]). List ranking can be used for list contractions as well. The *Euler tour technique* [TV85] can be used to optimally perform depth first search on trees, and can be applied to perform various other operations on trees optimally. Specifically, given an unrooted tree, one can obtain an oriented (rooted) tree in $O(\log n)$ time by using this technique. Hagerup [Hage90] has presented algorithms to find the connected components and a spanning forest of a planar graph in $O(\log n)$ time using $O(n/\log n)$ processors on a CRCW PRAM[2]. We make use of these algorithms extensively.

An undirected graph is *d-bounded* if the degree of each of its vertices is bounded by d. An *orientation* of $G = (V, E)$ is a directed graph with each edge $(u, v) \in E$ replaced by a directed edge $u \to v$ or $v \to u$; i.e., we assign a direction to each edge. If the in-degree of every vertex of the orientation is bounded by a constant d, then we will say that it is a *d-bounded orientation* of G. Clearly any undirected graph of maximum degree d has a d-bounded acyclic orientation. It is also known that every planar graph has a 6-bounded

[2] As Hagerup points out, his algorithm actually applies to a larger class of graphs called linearly contractible graphs, which we will discuss in Section 5.

acyclic orientation, and that one can be found in linear time sequentially and optimally in $O(\log n \log^* n)$ time in parallel on an EREW PRAM [CE91].

In the adjacency list representation of an undirected graph, there are two list nodes corresponding to each edge: one on the adjacency list of each endpoint. Call these two nodes *partners* of each other. The *cross-linked* adjacency list representation of an undirected graph has links between partners. We assume that graphs or directed graphs are represented as an array of cross-linked adjacency lists; this assumption of the existence of cross-links is common in parallel algorithms. (An adjacency list with cross links can be constructed by means of a sorting of the undirected edges [HCD87].) Some of our algorithms require processors to be able to write to the same shared memory location simultaneously. The write conflicts in these cases are resolved arbitrarily, where any one of the processors performing a concurrent write succeeds (i.e., we use the ARBITRARY CRCW PRAM model to resolve conflicts during concurrent writes). Without loss of generality, we assume that the vertices are numbered from 1 to n.

3. Maximal Matching in a Forest

We assume that trees are represented by adjacency lists. In particular, we assume that an oriented forest is represented by providing an array of lists of the children of each node.

Under these assumptions, maximal matching in an oriented tree can be computed as follows. First form the vertices into a set of linked lists by setting Link[x] to nil when x is a leaf, and Link[x] to point to any child of x when x is an internal node. Using list ranking we can set, for each x, Dist[x] to be the number of nodes preceding x on its linked list; i.e., we number the nodes in each list $0, 1, 2, \ldots$. Now, for each x, we place the edge (x,Link[x]) in the matching iff Dist[x] is even. This will certainly form a matching, since we are only selecting alternate edges from a set of disjoint paths. Moreover, it must be a maximal matching, since all internal nodes are endpoints of edges in the matching and any edge not in the matching has an internal node as an endpoint.

This approach easily generalizes to any undirected graph G without cycles, since we can first divide G into trees, orient each tree, and then apply the algorithm described above to each tree. This is summarized in Algorithm 1.

Algorithm 1: Maximal Matching in a Forest
Input: A Forest F

1. Divide F into trees and select a root for each tree.
2. Direct the edges of the trees away from the roots.
3. Let each internal node choose (arbitrarily) one of its outgoing edges, forming a set of paths.
4. List rank each path.
5. In each path, select alternate edges beginning with the first.

Lemma 1. Algorithm 1 can be implemented to perform maximal matching in an undirected forest in $O(\log n)$ time optimally on a CRCW PRAM. If the input is an oriented forest, we can perform the maximal matching in $O(\log n)$ time optimally on an EREW PRAM.

Proof: Step 1 can be done using Hagerup's connected component algorithm [Hage90] optimally in $O(\log n)$ time on a CRCW PRAM. Step 2 can be done in $O(\log n)$ time optimally on a CRCW PRAM using an Euler tour of the graph (see [KR88]). Steps 3–5 (which are all that is necessary if the input is an oriented forest) can be done in $O(\log n)$ time optimally on an EREW PRAM using an optimal list ranking algorithm (see [CV88] and [AM88]). ∎

4. Bounded Degree Graphs

We now consider maximal matching in a bounded degree graph.

Theorem 1. Algorithm 2 finds a maximal matching in a d-bounded graph on an EREW PRAM optimally in $O(\log n)$ time.

Algorithm 2: Maximal Matching in Bounded Degree Graphs

Input: A d-bounded graph G

1. $M = \emptyset$.
2. Orient the edges in such a way that each edge points from a higher numbered vertex to a lower numbered vertex.
3. Initialize all edges to be unmarked.
4. Repeat d times:
   - 4a. for each vertex, choose the first unmarked in-edge (if any) on the vertex's adjacency list; let the directed subgraph formed by the chosen edges be called G'.
   - 4b. use steps 3-5 of Algorithm 1 to find a maximal matching M' for G'; set $M \leftarrow M \cup M'$.
   - 4c. mark all edges which are incident upon endpoints of edges in M'.

Proof: The marks can be interpreted as indicating that an edge is ineligible for the matching. The final result must be a matching, since each time we add a set M' of edges to the matching we mark as ineligible all edges which share a vertex with an edge in M'. To see that it is maximal, first note that we do eventually mark all edges, since each adjacency list has at most d in-edges, and each iteration of step 4 causes at least one of them to be marked if any remain. Then since each marked edge is either placed into M or is ineligible since it shares a vertex with an edge in M, the matching must be maximal.

Note that the orientation formed in step 2 is acyclic. Hence, after step 4a, since each vertex chooses one in-edge, the resulting graph G' is a directed out-forest (where the edges are directed away from the roots).

Assume that we have oriented the graph G by simply placing a flag in each node of the adjacency lists to indicate whether it corresponds to an in-edge or out-edge. In Step 4a, after selecting the edges that form G', we wish to obtain an adjacency list representation for G', i.e., we wish to obtain an adjacency list of out-edges for each vertex from the selected in-edges. This can be done easily and quickly in the case of bounded degree graphs as follows.

Let H be a copy of G such that every edge $e = (v, w)$ in the adjacency lists of G and its copy in the adjacency lists of H are linked to one another. (This enables us to access an edge in H from the same edge in G and vice versa in constant time.) When an in-edge is selected in G in Step 4a, we flag the partner of the copy of the edge in H (notice that the partner is an out-edge). By contracting the adjacency lists of H to retain all the flagged edges, we obtain the required representation for G'. Since the adjacency lists of H are bounded, the required adjacency representation for G' can be obtained in constant time on an EREW PRAM using $O(n)$ processors.

For Step 4c, we first flag all vertices which are endpoints of M', then mark all nodes on the adjacency lists of flagged vertices, and finally propagate this information to partners via the cross-links. Thus this step can also be done in constant time using $O(n)$ processors because of the degree bound.

By Lemma 1, step 4b can be done optimally on an EREW in $O(\log n)$ time. Hence Algorithm 2 can be implemented to run in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. ∎

5. Maximal Matching in K_p-Minor Free Graphs

In this section we describe algorithms to perform maximal matching in graphs which are K_p-minor free for fixed p. This covers a number of interesting classes of graphs; for example, for $p \geq 5$, a graph of genus g is K_p-minor-free if $g < \lceil (p-3)(p-4)/12 \rceil$ (see [CL86, Section 4.4, Theorem 4.26]).

Following Hagerup [Hage90], we say a class $\mathcal{G}$ of graphs is *linearly contractible* if for all $G = (V, E)$ in $\mathcal{G}$,

a) $|E| = O(|V|)$, and

b) every minor of G is also in $\mathcal{G}$.

In particular, if (a) can be replaced by $|E| \leq c|V|$, we will say $\mathcal{G}$ is *linearly contractible with parameter c*.

It is known that every graph with $2^{p-3}|V|$ or more edges has a K_p minor (see [Boll78, Chapter 7, Theorem 1.14]), so if $G = (V, E)$ is a K_p-minor-free graph we must have $|E| < 2^{p-3}|V|$. Thus the class of K_p-minor-free graphs is linearly contractible, for fixed p.

5.1. A CRCW PRAM algorithm

We begin with the following observation.

Lemma 2. If G is a connected K_p-minor-free graph, then the graph induced by the leaves of any spanning tree of G is K_{p-1}-minor-free.

Proof: Let T be a spanning tree of G, and let G' be the graph induced by the leaves of T. Suppose for a contradiction that K_{p-1} is a minor in G'. By contracting all internal nodes of T into a single vertex we obtain K_p as a minor in G, giving the desired contradiction. ∎

Note that it follows immediately that if G is a K_p-minor-free graph, then the graph induced by the leaves of any spanning forest of G is K_{p-1}-minor-free.

Theorem 2. For fixed p, maximal matching in a K_p-minor-free graph G can be performed in $O(\log n)$ time using $O(n/\log n)$ processors on a CRCW PRAM.

Proof: Generate a sequence of graphs $G_0 = G, G_1, \ldots, G_t$, where each G_i is obtained from G_{i-1} by

i) computing a spanning forest F_{i-1},

ii) computing a maximal matching M_{i-1} in F_{i-1} using Algorithm 1, and then

iii) letting G_i be the subgraph of G_{i-1} induced by the unmatched vertices.

Since only a subset of leaves remain after a maximal matching is computed for a forest, from Lemma 2 we know that G_i is K_{p-i}-minor-free. After $t = p - 3$ iterations, the remaining graph G_t is K_3-minor-free, which implies that G_t must be a forest. Algorithm 1 can then be applied to obtain a maximal matching of this forest.

Since G and its successors are K_p-minor-free, and hence linearly contractible, Hagerup's algorithm can be used to obtain a spanning forest and label connected components in $O(\log n)$ time using $O(n/\log n)$ processors on a CRCW PRAM. Each application of Algorithm 1 also requires $O(\log n)$ time using $O(n/\log n)$ processors. After computing M_{i-1} in the forest F_{i-1}, we mark all the vertices (and edges) in G_{i-1} that are also in M_{i-1}. We then compute G_i from G_{i-1} by contracting every adjacency list using list ranking. This process of computing G_i can be done optimally in $O(\log n)$ time.

The number of iterations is $t = O(p)$, which is $O(1)$ since we assume p is fixed. Hence the whole procedure operates within the stated resource bounds. ■

5.2. An EREW PRAM algorithm

We now obtain a slightly slower algorithm that can be run optimally on an EREW PRAM. First we show how a graph G which is K_p-minor-free and hence linearly contractible can be oriented such that the in-degree of any node in the graph is bounded. Algorithm 3 is a straightforward generalization of Chrobak and Eppstein's [CE91] optimal $O(\log n \log^* n)$ EREW PRAM algorithm to find 6-bounded acyclic orientations for planar graphs, which, in turn, is very similar to the parallel 5-coloring algorithm for planar graphs presented in [HCD87].

Theorem 3. Given an undirected graph $G = (V, E)$ which is linearly contractible with parameter c, Algorithm 3 will orient it to be acyclic and have in-degree bounded by $4c$. Moreover, this algorithm can be implemented to run optimally in $O(\log n \log^* n)$ time on an EREW PRAM.

Proof: The computation is in $O(\log n)$ phases. In phase i, we find a set S_i of vertices of degree at most $4c$. Now we construct a graph $H_i = (S_i, F_i)$, where $(u, v) \in F_i$ if either $(u, v) \in E$ or u and v have a common neighbor x such that the edges (u, x) and (v, x) are consecutive in the adjacency list of x. We then compute a maximal independent set V_i in H_i. Since H_i is a bounded degree graph, $|V_i| = \Omega(|S_i|)$ (see [HCD87], and [GPS87]). Selecting independent vertices in this manner enables us to perform Step 2d in constant time as the list contraction process involved in obtaining G_{i+1} can be performed in constant time.

The correctness of the algorithm can be seen easily. Note that since each V_i is an independent set, all edges will run between vertices with different labels, and hence the algorithm produces an acyclic graph. Now if v is an arbitrary vertex with label i, then

Algorithm 3: Orienting a Linearly Contractible Graph to Have Bounded In-Degree
Input: An undirected graph G which is linearly contractible with parameter c

1. $G_0 \leftarrow G$; $i \leftarrow 0$.
2. While G_i contains any vertices do
   - 2a. Let S_i be the set of all vertices of degree less than $4c$ in G_i.
   - 2b. Let V_i be a maximal set of independent vertices among S_i, such that if $v, u \in V_i$, and $(v, x), (u, x) \in E$, then v and u are not consecutive elements in the adjacency list of x.
   - 2c. Assign the label i to each vertex in V_i.
   - 2d. Eliminate V_i from G_i to obtain G_{i+1}.
   - 2e. Increment i.
3. Orient the edges in such a way that each edge points from a higher numbered vertex to a lower numbered vertex; Obtain adjacency lists consisting of just in-edges.

$v \in V_i$ and all in-edges of v must have been present in G_i, so since all vertices in V_i had degree at most $4c$, the in-degree of v is bounded by $4c$.

Next we show that there are only $O(\log n)$ iterations in step 2. Let n_i be the number of vertices in G_i. First note that the set S_i must contain at least $n_i/2$ vertices, since the total of the degrees of the vertices in G_i is at most $2cn_i$. Thus at each iteration we remove a constant fraction of the vertices (as $|V_i| = \Omega(n_i)$), so it takes only $O(\log n)$ iterations to remove them all.

Finally, we show that the algorithm can be implemented to run within the given bounds. Step 1 is straightforward. Aside from 2b, the substeps within step 2 can be done in constant time with $O(n)$ work, since we only need constant time to scan adjacency lists of length bounded by $4c$.

Each execution of Step 2b can be done in $O(\log^* n)$ time using $O(n)$ processors or in $(\log n)$ time using $O(n/\log n)$ processors ([HCD87], [GPS87]). We use the second method for the first $O(\log^* n)$ iterations. The total work done during these $O(\log^* n)$ iterations is $O(n)$ as the size of the graph decreases by a constant factor each time. We then switch to the faster method. Again, the total work done is $O(n)$ as the size of the graph decreases by a constant factor after each iteration. The total time taken is $O(\log n \log^* n)$.

In order to obtain an optimal algorithm, we use Brent's theorem [Bren74] and reduce the number of processors. Reduction in the number of processors requires a scheduling method to balance the work load among the smaller number of processors. We can achieve this by keeping a list of remaining vertices and edges and by compacting the list to get rid of deleted vertices and edges. The list compaction can be done using a prefix computation [LF80], which takes $O(\log n)$ time and does $O(n)$ work. We perform the compaction at appropriate intervals to achieve the $O(\log n \log^* n)$ time using $O(n/\log n \log^* n)$ processors [CE91]. ■

When we orient an input graph G with Algorithm 3, we obtain a directed graph G' with the in-degree of each vertex bounded by a constant (namely, $4c$). This does not enable us to apply the result of Theorem 1 directly, since that theorem requires that the degree of the undirected graph be bounded by a constant. Fortunately, a variation of Algorithm 2 can be used to find a maximal matching for G; we omit step 2 and take extra care in the implementation of step 4. Step 2 can be omitted since the graph has already been oriented as needed, and the desired adjacency in-lists have been obtained. Most of step 4 can readily be done optimally. After performing Step 4a, as in the bounded-degree case of Section 4, we obtain an adjacency list consisting of out-edges from the selected in-edges. Unlike in the bounded-degree case, the adjacency lists of G may not be of bounded length; but we can still perform the necessary operations (such as flagging edges and compacting lists) within the resource bounds of $O(\log n)$ time and $O(n/\log n)$ processors. In Step 4c, some care is needed when we mark all edges incident with the endpoints of edges of M'.

Assume that when we create the oriented graph (call it H) in Algorithm 3, we keep a copy of the original graph G. Assume also that whenever we add a directed edge $e = (v, w)$ (e is directed from v to w) to the adjacency list of w in H, we let the corresponding edge on the adjacency list of w in G point to e. We can now mark all edges in H which are incident with endpoints of M' as follows.

i) using list ranking, mark all edges on the adjacency lists in G of all vertices which are endpoints of M'.

ii) using the cross links, mark the partners of each marked edge in G.

iii) for each marked edge in G which points to an edge e in H, mark e.

This process can be done optimally in $O(\log n)$ time, and step 4 is iterated only $O(1)$ times. Thus we obtain the following:

Corollary. Maximal matching can be performed on any linearly contractible class of undirected graphs optimally in $O(\log n \log^* n)$ time on an EREW PRAM.

6. Conclusion

We have presented fast parallel algorithms for maximal matching for a large class of sparse graphs. Using a CRCW PRAM, we can find a maximal matching for these sparse graphs in $O(\log n)$ time. Using an EREW PRAM, we can find a maximal matching in $O(\log n)$ time for graphs of bounded degree, and in $O(\log n \log^* n)$ time for the K_p-minor-free graphs. All these algorithms are optimal, i.e., they require only $O(n)$ work. For general graphs the previously best known algorithm was that of Israeli and Shiloach [IS86] which used $O(\log^3 n)$ time and $O(m \log^3 n)$ work. Even for planar graphs, which are a small subset of the class we consider, the previously best known algorithm on a CRCW [GPS87] used $O(\log n \log^* n)$ time with $O(n)$ processors.

Acknowledgements

We are grateful to Dan Hirschberg for helpful comments on an earlier draft.

References

[AM88] R. J. Anderson and G. L. Miller. Deterministic Parallel List Ranking. *Algorithmica* 6 (6), pages 859–868, 1991.

[Boll78] B. Bollobás. *Extremal Graph Theory.* Academic Press, London, 1978.

[Bren74] R. P. Brent. The Parallel Evaluation of General Arithmetic Expressions. *J. Assoc. Comput. Mach.* 21, pages 201–206, 1974.

[CL86] G. Chartrand and L. Lesniak. *Graphs & Digraphs.* The Wadsworth & Brooks/Cole Mathematics Series. Wadsworth, Inc., Monterey, California, second edition, 1986.

[CE91] M. Chrobak and D. Eppstein. Planar Orientations with Low Out-degree and Compaction of Adjacency Matrices. *Theoretical Computer Science*, 86:243-266,1991.

[CY88] M. Chrobak and M. Yung. Fast Parallel and Sequential Algorithms for Edge-Coloring Planar Graphs (extended abstract). In J. H. Reif, editor, *VLSI Algorithms and Architectures: 3rd Aegean Workshoop on Computing*, pages 11–23, Corfu, Greece, June/July 1988. Springer-Verlag. Lecture Notes in Computer Science 319.

[CV88] R. Cole and U. Vishkin. Optimal Parallel Algorithms for Expression Tree Evaluation and List Ranking. In J. H. Reif, editor, *VLSI Algorithms and Architectures: 3rd Aegean Workshoop on Computing*, pages 91–100, Corfu, Greece, June/July 1988. Springer-Verlag. Lecture Notes in Computer Science 319.

[Edmo65] J. Edmonds. Paths, Trees and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Gali86] Z. Galil. Efficient Algorithms for Finding Maximum Matching in Graphs. *ACM Computing Surveys*, 18(1):23–38, March 1986.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

[GPS87] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel Symmetry-breaking in Sparse Graphs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 315–324, 1987.

[Hage90] T. Hagerup. Optimal Parallel Algorithms on Planar Graphs. *Information and Computation*, 84:71–96, 1990.

[HCD87] T. Hagerup, M. Chrobak, and K. Diks. Optimal 5-coloring of Planar Graphs. *SIAM Journal on Computing*, 18:288–300, 1989.

[HK73] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

[IS86] A. Israeli and Y. Shiloach. An Improved Algorithm for Maximal Matching. *Information Processing Letters*, 22(2):57–60, January 1986.

[KR88] R. M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, Chapter 17, pages 869–941, Elsevier, Amsterdam, 1990.

[KUW86] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a Perfect Matching is in Random NC. *Combinatorica*, 6(1):35–48, 1986.

[LF80] R. E. Ladner and M. J. Fischer. Parallel Prefix Computation. *J. Assoc. Comput. Mach.* 27, pages 831–838, 1980.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as Easy as Matrix Inversion. *Combinatorica*, 7(1):105–113, 1987.

[TV85] R. E. Tarjan and U. Vishkin. An Efficient Parallel Biconnectivity Algorithm. *SIAM Journal on Computing*, 14(4):862–874, November 1985.