A Parallel Algorithm for Finding Maximal Independent Sets in Planar Graphs[†]

N. Dadoun
D.G. Kirkpatrick

Technical Report 87-15
June 1987

Abstract

The problem of constructing parallel algorithms for finding Maximal Independent Sets in graphs has received considerable attention. In the case that the given graph is planar, the simple efficient parallel algorithm described here may be employed. The method relies on an efficient parallel algorithm for constructing large independent sets in bounded degree graphs. This is accomplished by a simple reduction to the same problem for lists.

Using a linear number of EREW processors, the algorithm identifies a maximal independent set in an arbitrary planar graph in $O(\log n \log^* n)$ parallel time. A randomized version of the algorithm runs in $O(\log n)$ expected parallel time.

[†] A preliminary version of some of the results in this report was presented in [DaK1].

I Introduction

A set **I** of vertices in a graph **G** = (**V**, **E**) is *independent* if no pair of vertices in **I** is connected by an edge. An independent set is *maximal* if it is not a proper subset of any other independent set (equivalently every vertex in **V** - **I** is incident with at least one vertex in **I**). The *Maximal Independent Set Problem* MISP(n) is defined as the problem of identifying, for an arbitrary input graph with n vertices, a maximal independent set of vertices.

This paper presents a method for solving the MISP(n) for planar graphs using a parallel processing model of computation. Note that by Euler's theorem the size of the edge set of a planar graph is linear in the size of the vertex set and hence the single parameter n is a true reflection of the size of the entire graph. Our algorithm assumes as input a graph **G** represented as an array of vertices **V** along with an array of edges **E**. Each vertex v in **V** has a pointer to a ring of edges incident with v.

Our algorithms are described for a Single Instruction/Multiple Data (SIMD) shared memory PRAM using an exclusive read/exclusive write (EREW) memory model. We are interested in the case that the number of processors is linear in the input size n; so, without loss of generality, we assume that each vertex and edge has associated with it a dedicated processor. Each processor has a distinct identifier which can be used to make local decisions. The processor identifier of a processor assigned to a vertex (respectively edge) will also be referred to as the vertex (respectively edge) number.

We say that a problem of size n has parallel complexity $f(n)$ if it can be solved in $O(f(n))$ parallel time using $O(n)$ processors as described above. Our interest lies primarily in understanding the asymptotic complexity of the maximal independent set problem; no attempt is made here to optimize the constants involved.

Finding a maximal independent set (MIS) of vertices in an arbitrary graph is a problem which has a simple and efficient sequential solution but which has not yielded easily to parallel solutions. Karp and Wigderson [KW] were the first to provide a polylogarithmic parallel solution. Their O($log^4 n$) parallel time algorithm uses O($n^3/log^3(n)$) exclusive read/exclusive write (EREW) processors. Luby [L] improved this result to show that a MIS can be found in O($log^2 n$) parallel time using O(n^3) EREW processors. Although planarity of the underlying graph does not seem to be something which is easily exploited in the sequential setting, it turns out to lead to both simpler and more efficient solutions in the parallel case.

Our algorithm is based on the following variant of the planar MISP. Informally, the *Fractional Independent Set Problem* FISP(n) is defined as the problem of identifying for an arbitrary planar graph **G** with n vertices an independent set **I** of "low-degree" vertices in **G** whose size $|\mathbf{I}|$ is some fixed fraction of n.

Section II, through a series of reductions, describes a solution for the fractional independent set problem which runs in O($log^* n$) (respectively O(1) expected) parallel time using a deterministic (respectively randomized) algorithm.

Section III gives a reduction of the planar MISP to O($log\ n$) instances of the fractional independent set problem. This combined with the results of section II produces an O($log\ n\ log^* n$) (respectively O($log\ n$) expected) parallel time deterministic (respectively randomized) algorithm for the Maximal Independent Set problem for planar graphs.

II The Fractional Independent Set Problem

It follows from Euler's theorem that the average degree of vertices in a planar graph is less than 6 and so less than half of the vertices have degree exceeding 11 (cf. [K]). A vertex is said to be *low degree* if it has degree bounded by some constant b. By the argument above, every planar graph has a subgraph of low degree vertices which forms a fixed fraction (based on b) of the graph [LM] [K] [EKA].

The *Fractional Independent Set Problem* FISP(n) is defined as the problem of identifying for an arbitrary planar graph **G** with n vertices an independent set **I** of low degree vertices such that $| \mathbf{I} | > n / c$ for some fixed constant c. Special cases of this problem, BD-FISP(n) and L-FISP(n), concern the restriction to *bounded degree graphs* (graphs composed entirely of low degree vertices), and *chains* or *list graphs* (digraphs whose vertices have in-degree and out-degree bounded by 1) respectively. We also use the notation MISP(n) and FISP(n) to refer to the parallel complexity of solving these respective problems.

It is possible to relate the parallel complexities of variants of the FISP by means of some straightforward reductions:

Lemma 1: FISP(n) $\leq$ O(1) + FISP-BD(n)

Proof: Since the low degree vertices form a fixed fraction of the vertices of a planar graph, it suffices to identify a subgraph of the input graph such that all vertices have degree at most b. In the following description, the edge 'near end' and 'far end' locations are used to avoid read/write conflicts.

Each vertex processor marks its vertex high degree (as a default). It then counts its incident edges up to a maximum of b + 1. Any processor which counted up to b + 1 edges sits out and the others mark themselves low-degree. Each low-degree vertex marks the 'far end' of its incident edges High Degree. It then marks the 'near end' of its incident edges Low Degree. If then reads the 'far end' of its incident edges and removes from its edge list the ones which are marked High Degree. In this way, all low degree vertices identify themselves, their low degree neighbours and the resulting induced graph in O(1) parallel time.

The following pseudo-code procedure, which is executed in parallel by each of the vertex processors, describes in more detail the procedure to identify the graph induced on the low degree vertices.

```
procedure LowDegreeSubgraph;

begin
    Mark vertex high-degree;
    degree := 0;
    test_edge := first_edge;
    counted_all := false;

    (* Count edges to identify low-degree vertices. *)

    for k := 1 to (b + 1) do
        if (not counted_all) then
            begin
                degree := degree + 1;
                test_edge := test_edge.next;
                counted_all := (test_edge = first_edge)
            end;

    (* Remove edges which are incident on high-degree vertices. *)

    if degree ≤ b then
        begin
            Mark vertex low-degree;
            for j := 1 to b do if (edge j <> null) then Mark 'far end' of edge j high-degree;
            for j := 1 to b do if (edge j <> null) then Mark 'near end' of edge j low-degree;
            for j := 1 to b do if (edge j <> null) then
                if 'far end' of edge j is marked high-degree
                    then remove edge j from edge list
        end
end.          •
```

Lemma 2: BD-FISP(n) ≤ O(1) + c L-FISP(n)

Proof: Assume we are given as input a graph G each of whose vertices has degree at most b. Each vertex is assigned a processor. We first show how the edge set E of the graph G can be decomposed, in constant time, into a constant number $t < 2b$ of sets E_i ($1 \leq i \leq t$) where each set E_i defines a list graph.

The sets E_i ($1 \leq i \leq t$) will be formed in t rounds. A set of chains is identified in parallel by allowing each vertex to determine at most one incoming edge and at most one outgoing edge from its remaining incident edges. As an edge is chosen, it is marked as belonging to chain i and is removed from consideration. Each round is divided into b subrounds. In a subround, a vertex which has not yet had a proposal accepted proposes to a new neighbour (if one is available). If a vertex receives any proposals, it will accept exactly one and ignore all others. A vertex has all of its proposals ignored in a round only if all of its neighbours have accepted other proposals in this round. Hence after 2b rounds, all of a vertex's neighbours must have accepted its proposal.

We then find a Fractional Independent set in E_1 and mark them as survivors. Among the set of survivors in E_i we find a Fractional Independent set in E_{i+1}. The set of survivors in E_t will form a fixed fraction of the vertices of G. Therefore a constant number of iterations of the Fractional Independent Set Problem for list graphs suffices to solve the Fractional Independent Set Problem for bounded degree graphs.

The following pseudo-code procedure, which is executed in parallel by each of the vertex processors, describes in more detail the procedure to decompose a degree bounded graph into a set of list graphs.

```
procedure DecomposeIntoChains;
begin

    (* Each iteration identifies 1 outgoing edge and 1 incoming edge.  *)

    for i := 1 to t do
        begin
            in_mated := false;
            out_mated := false;

            for j := 1 to b  do
                begin

                    (* Propose to a neighbour *)

                    if (not out_mated)  and (edge j <> null) then
                        Mark 'Far End' of edge j propose;

                    (* Check proposals from neighbours *)

                    for k:= 1 to b do
                        if(not in_mated) and(edge k <> null) and
                            ('near end' of edge k is marked propose) then
                             begin
                                Mark 'near end' of edge k accept;
                                in_mated := true;
                                in_edge := k
                             end;

                    (* See if proposal was accepted *)

                    if(not out_mated) and (edge j <> null) and
                        ('far end' of edge j is marked accept) then
                         begin
                            out_mated := true;
                            out_edge := j
                         end
                end;

            (* Record incident edges for chain i *)

            if in_mated then
                begin
                    Mark 'near end' of edge in_edge in-chain (i);
                    Remove edge in_edge from current edge ring
                end;

            if out_mated then
                begin
                    Mark 'near end' of edge out_edge out-chain (i);
                    Remove edge out_edge from current edge ring
                end;
        end
end.    •
```

Note that the Fractional Independent Set Problem for n vertex list graphs can be solved by standard list ranking in O(*log n*) parallel time using O(n) EREW processors [W]. However, full list ranking is unnecessary.

Lemma 3: L-FISP(n) can be solved in O($log^* n$) parallel time using a deterministic algorithm.

Proof: Cole and Vishkin [CV] define an *r - ruling set* on a list graph L on n vertices to be a subset U of the vertices of L such that: i) No two vertices of U are adjacent; and ii) For each vertex v in L there is a directed path from v to some vertex in U whose edge length is at most r. Cole and Vishkin show how to find a 2-ruling set in O($log^* n$) time using a technique which they call deterministic coin tossing. Note that a 2-ruling set is a fractional independent set for its list. •

It is straightforward to modify Cole and Vishkin's algorithm to find a fractional independent set among any subset of the vertices of a list.

Lemma 4: L-FISP(n) can be solved with probability 1 - O(c^n), for some $c < 1$, in O(1) parallel time using a randomized algorithm.

Proof: Assign a processor to each vertex of the list. Each processor flips a 0 or a 1with equal probability. A vertex is chosen if its processor flips a 1 and either it has no successor or its successor flips a 0. With probability at least 1/4 an arbitrary vertex is chosen. However, these probabilities are not independent. Nevertheless, every second list element is chosen independently with a probability of at least 1/4. Thus applying the Chernoff bound (cf. [PB, p. 464]), we find that the probability that fewer than 1/8 of the even positioned list elements are chosen is at most c^n, where $c < 0.98$. •

III Maximal Independent Sets in Planar Graphs

The algorithm to find a MIS in a planar graph is based on the fact that a fractional independent set can be found in a low-degree graph quickly. A (somewhat idealized) conceptual description of the algorithm follows:

For planar graph G = (V, E):

```
begin
    $H \leftarrow V$;
    $I \leftarrow \emptyset$;
    while $|H| \neq 0$ do
    begin
        [a] Identify a Fractional Independent Set M of H;
        [b] $I \leftarrow I \cup M$;
        [c] $H \leftarrow H - (M \cup \Gamma_H(M))$;
    end;
end.
```

$\Gamma_H(M)$ is the *neighbourhood* of M in H, the set of vertices in H which are adjacent to a vertex of M and all of their incident edges. Since each iteration of the while loop reduces the size of the set H by a fixed fraction, there are $O(\log n)$ iterations. Note that this exploits the fact that any subgraph of a planar graph is still a planar graph.

The naive way of implementing the algorithm would be to logically complete an iteration - remove entirely the identified fractional independent set along with its neighbourhood - before proceeding with the next iteration. However, some of the neighbourhood vertices may be high-degree and hence on an EREW model of computation it may require logarithmic time to notify all of an eliminated vertex's neighbours that it has been removed. Each round would then take $O(\log n)$ parallel time resulting in an $O(\log^2 n)$ parallel time MIS algorithm.

A more efficient algorithm, which conforms to the EREW model, uses the following more sophisticated strategy for an iteration. (Note that for simplicity, we consider the identification of a fractional independent set to be an indivisible operation.)

A vertex is labelled or unlabelled: if it is labelled then it is labelled chosen or unchosen and hence its membership in the independent set has been determined; if it is unlabelled then it is still under consideration for being a member of the independent set. Initially, all vertices are unlabelled.

An edge is unmarked or marked: if it is unmarked then both its incident vertices remain unlabelled; if it is marked then at least one of its incident vertices is labelled. A marked edge is satisfied or forbidden: if it is satisfied then one of its incident vertices has been labelled unchosen (a neighbourhood vertex) hence the constraint this edge represents has been satisfied; if it is forbidden then one of its incident vertices has a vertex neighbour which been labelled chosen (a member of the independent set) hence the remaining unlabelled vertex is to be labelled unchosen. This latter case of an edge being marked forbidden may occur as a result of a vertex being labelled chosen or as a result of an edge ring compression in the cleanup operation described below.

An iteration proceeds as follows: A fractional independent set of vertices M is identified from H. Each vertex v in M is low degree, hence in constant time v can examine and/or mark all its incident edges (without read/write conflicts since M is an independent set). If any of v's incident edges are marked as being incident on a chosen vertex (ie. v is a neighbourhood vertex from a previous selection) then v labels itself unchosen, removes itself from H and marks its incident edges satisfied (ie. their associated independence constraint is "satisfied" and adjacent vertices can remove them without consequence). If none of v's incident edges are marked as being incident on a chosen vertex then v labels itself chosen, removes itself from H and marks its incident edges forbidden (ie. its adjacent vertices are neighbourhood vertices and their future choice for membership in the independent set is "forbidden"). At this point, each vertex in H has within its edge ring a number of marked edges and a number of unmarked edges. The marked edges are those scheduled for removal. A vertex is selected as a low degree vertex only when its edge ring (including marked edges) falls below a certain size. Thus the final operation within an iteration is to remove at least half of the remaining marked edges so that there will be some low degree candidate vertices for the FIS operation of the next iteration.

The algorithm incorporating the iteration strategy outlined above follows:

For planar graph G = (V, E):

```
begin
    H ← V;
    I ← ∅;
    while |H| ≠ 0 do
    begin
        [a] Identify a Fractional Independent Set M of H;
        [b] Label eligible vertices in M "chosen", remove them from H,
            and mark their incident edges "forbidden";
        [c] Label ineligible vertices in M "unchosen", remove them from H,
            and mark their incident edges "satisfied";
        [d] Cleanup by removing at least half the remaining marked edges;
    end;
end.
```

The steps [a], [b], and [c] are sufficiently described above. The step [d] cleanup operation requires elaboration. Each unlabelled vertex has a list of incident edge pointers; some of these edges may be marked "forbidden", some of these edges may be marked "satisfied" and the rest are unmarked. There may be many consecutive marked edges. We wish to remove at least half of the marked edges without increasing the cost of an iteration while conforming to the EREW restriction.

We can remove a fixed fraction of the marked edges by identifying a fractional independent set of the marked edges within each edge list. This can be done using either the deterministic algorithm of Lemma 3 (finding a 2-ruling set [CV]) or the randomized algorithm of Lemma 4 depending on which algorithm is used in step [a].

Each edge e checks its successor f for membership in the fractional independent set. If f is in the fractional independent set then e adjusts its pointer around f. If f is marked forbidden (ie. f's far vertex is labelled chosen and hence f's near vertex must be labelled unchosen) then e must retain this information by marking its (near) end forbidden. Since a fixed fraction of the marked edges are removed in this fashion, repeating this operation some constant number of times will reduce the number of marked edges by one half.

The dominant cost within each iteration is the fractional independent set operation used in steps [a] and [d]. It remains to show that we can guarantee that there will still be a total of O(*log n*) rounds.

Each round eliminates a fixed fraction of the remaining vertices. It is possible that an unlabelled vertex has a large number of neighbours eliminated within a given round. That is, a vertex which becomes low degree within a particular round may not discover that it has become low degree for several rounds. However, using the marking/removal strategy described above, it is a straightforward inductive exercise to show that there are O(*log n*) rounds in total.

Specifically, if v_i denotes the number of vertices remaining at the end of round i ($v_0 = n$) and e_i (respectively f_i) denotes the number of unmarked (respectively marked) edges remaining at the end of round i, then

(1) $e_i < 3v_i$ (since the unmarked edges form a planar graph on the active vertices) and,

(2) $f_{i+1} \le e_i + f_i / 2$ (since the newly marked edges number at most e_i and the old marked edges are reduced in number by at least a factor of 1/2 in the cleanup operation).

If we choose low degree to mean degree less than 25 and let $\alpha = 49 / 50$, then

Lemma 5: For all $i \ge 1$,

i) $v_i \le \alpha^{i-1} n$,

ii) $e_i \le 3\alpha^{i-1} n$ and,

iii) $f_i \le (6\alpha^{i-1} n) / (2\alpha - 1)$

Proof (by induction on i):

The basis of the induction is clear. Suppose the result is true for $i < k$. By observations (1) and (2) above, it suffices to show that $v_{k+1} \le \alpha^k n$. Suppose that $v_k > \alpha^k n$. (otherwise the result is immediate). Since $e_k \leq 3\alpha^{k-1} n$ and $f_k \leq (6\alpha^{k-1}n) / (2\alpha - 1)$, it follows that the average degree of the v_k active vertices at the end of round k is less than $12 / (2\alpha - 1)$. Hence, at least half of these vertices have degree less than 25. It follows that at least $v_k / 50 \geq \alpha^k n / 50$ vertices are chosen in round k+1. Thus

$$\begin{aligned} v_{k+1} &\leq v_k (49 / 50) \\ &\leq \alpha^k n \end{aligned}$$ •

Since each round is dominated by the cost of constructing a fractional independent set in a bounded degree graph, it follows that:

Theorem 1: A maximal independent set of an *n*-vertex planar (planar or bounded degree) graph can be constructed in O(*log n log*n*) (respectively O(*log n*) expected) parallel time using a deterministic (respectively randomized) algorithm on O(*n*) EREW processors.

IV Discussion

We have exploited simple properties of planar graphs to produce a simple and efficient parallel algorithm for finding a Maximal Independent Set in a planar graph. In particular, the reduction in the number of processors from that required for Luby's more general algorithm [L] may make this algorithm practical to implement and to use as a subroutine for other related tasks. It is not clear how to reduce the processor requirement further; the Cole and Vishkin O(*log*n*) parallel time 2-ruling set algorithm [CV] and the randomized L-FISP result both require O(*n*) processors.

We have used techniques similar to those used here in a geometric setting [DaK2]. The algorithm for finding Fractional Independent Sets can be applied to the parallel construction of Subdivision Hierarchies for fast sequential subdivision search and to the construction of Polyhedral Hierarchies for solving a number of spatial query problems and constructing three dimensional convex hulls.

Very recently we have become aware of the related work of Chrobak, Diks, and Hagerup [CDH]. Though their focus is on parallel algorithms for graph colouring, their techniques are very similar to those presented here and we expect that they could be applied equally successfully in our settings.

Acknowledgement

This work was supported in part by the National Sciences and Engineering Research Council of Canada, grant A3583.

References

[CDH] Chrobak, M., Diks, K., and Hagerup, T., "Parallel 5-Colouring of Planar Graphs", Preprint (1987).

[CV] Cole, R. and Vishkin, U., "Deterministic Coin Tossing and Accelerating Cascades: Micro and Macro Techniques for Designing Parallel Algorithms", *Proc. of the 18th ACM Symposium on Theory of Computing* (1986), pp. 206-219.

[DaK1] Dadoun, N. and Kirkpatrick, D.G., "Parallel Processing for Efficient Subdivision Search", *Proc. of the 3rd ACM Symposium on Computational Geometry* (1987).

[DaK2] Dadoun, N. and Kirkpatrick, D.G., "Parallel Construction of Subdivision Hierarchies", To Appear *JCSS*.

[EKA] Edahiro, M., Kokubo, I. and Asano, T., "A New Point-Location Algorithm and Its Practical Efficiency - Comparison with Existing Algorithms", *ACM Transactions on Graphics* 3, 2 (1984), pp. 86-109.

[KW] Karp, R.M. and Wigderson, A., "A Fast Parallel Algorithm for the Maximal Independent Set Problem", *Proc. of the 16th ACM Symposium on Theory of Computing* (1984), pp. 266-272.

[K] Kirkpatrick, D.G., "Optimal Search In Planar Subdivisions" *SIAM Journal of Computing* 12,1 (1983), pp. 28-35.

[LM] Lipton, R.J., and Miller, R.E., "A Batching Method for Coloring Planar Graphs", *Information Processing Letters* 7,4 (1978), pp. 185-188.

[L] Luby, M., "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *Proc. of the 17th ACM Symposium on Theory of Computing* (1985), pp. 1-10.

[PB] Purdom, P.W., and Brown, C.A., *The Analysis of Algorithms*, Holt, Rinehart and Winston, New York (1985).

[W] Wylie, J.C., "The Complexity of Parallel Computation", Technical Report TR 79-387, Dept. of Computer Science, Cornell University, Ithaca, New York, 1979.