Information Processing Letters 35 (1990) 261–267
North-Holland

28 August 1990

ON THE TIME-SPACE COMPLEXITY OF REACHABILITY QUERIES FOR PREPROCESSED GRAPHS

Lisa HELLERSTEIN *
UC Berkeley

Philip KLEIN **
Harvard University

Robert WILBER ***
AT&T Bell Labs

Communicated by D. Gries
Received 4 October 1989
Revised 2 March 1990

How much can preprocessing help in solving graph problems? In this paper, we consider the problem of reachability in a directed bipartite graph, and propose a model for evaluating the usefulness of preprocessing in solving this problem. We give tight bounds for restricted versions of the model that suggest that preprocessing is of limited utility.

Keywords: Computational complexity, preprocessing, graph reachability, time-space trade-offs

1. Introduction

The *directed reachability problem* is as follows: given an n-node graph G and a set S of nodes of G, determine the set of nodes T that are reachable from nodes of S. This problem can trivially be solved in time proportional to the number of edges in G, or $O(n^2)$ in the worst case. Moreover, this bound is tight to within a constant factor. If we are given G in advance, however, we could conceivably construct a representation of G that allows solution of the problem for any given S in much less time. We show that, for restricted models, this is not the case.

We prove our bounds for the special case in which G is a bipartite graph with all edges directed from one block of the bipartition to the other. Note that such a bipartite graph can represent the reachability relation of an arbitrary digraph. That is, given any n-node digraph, we can construct a $2n$-node directed bipartite graph in which the existence of a path in the digraph from v to w is represented by an edge from v to w' in the bipartite graph.

We investigate the tradeoff between the space required for a representation of G, and the worst-case time required to answer a query of the form "given S, find the neighbor set T". Our measure

* Research supported by NSF Grant CCR-8411954 and by the AT&T Bell Labs GRPW program.
** Research supported by the AT&T Bell Labs URP program, and by a fellowship from the Center for Intelligent Control Systems, with additional support from Air Force Contract AFOSR-86-0078, from PYI awarded to David Shmoys, with matching support from IBM and Sun Microsystems, and from ONR grant N00014-88-K-0243 and DARPA Grant N00039-88-C0113 at Harvard University.
*** Research supported by AT&T Bell Labs.

of time is that of Yao [7], namely the number of cells of memory that must be read to determine the output. The problem of determining the space-time tradeoffs for specific kinds of queries has been studied previously, e.g., [2–5,7–9]. The authors of [1] considered a related issue, the tradeoff between the time for preprocessing and the time for answering queries.

2. The problem

We consider the set $\mathscr{G}$ of bipartite graphs consisting of *input nodes* $x_1, \ldots, x_n$, *output nodes* $y_1, \ldots, y_n$, and edges between input and output nodes. Let $[n]$ denote $\{1, \ldots, n\}$. We may interpret a graph $G \in \mathscr{G}$ to be a subset $E(G) \subseteq [n] \times [n]$. Suppose we preprocess a graph G in $\mathscr{G}$. We investigate the time complexity of the following problem: for any *query* consisting of a subset S of input nodes, determine the *output set* T of output nodes adjacent to nodes of S in the graph G. Without preprocessing, the worst-case time complexity of this problem is clearly $\Theta(n^2)$.

One may interpret this problem as the problem of representing an $n \times n$ Boolean matrix A in such a way that for any Boolean n-vector x, the product Ax over the AND-OR semiring can be determined quickly.

3. Our model for preprocessing

A *representation* for $\mathscr{G}$ is a sequence $F = \langle f_1, \ldots, f_s \rangle$ of Boolean functions on $\mathscr{G}$, called "cells". For a given graph $G \in \mathscr{G}$, the values $f_i(G)$ are intended to give information about the graph G. We may also interpret the cells f_i as functions of n^2 Boolean variables corresponding to the possible edges of G, where a variable is 1 if the corresponding edge is present in G. For example, one trivial representation uses n^2 functions f_{ij}, where $f_{ij}(G)$ is 1 if the edge (x_i, y_j) is present in G, and 0 otherwise. To "probe" a cell is to determine its value for a specific (unknown) graph G. The *size* of a representation is the number s of cells.

A query scheme is, informally, a method for answering queries by probing cells of a representation. A scheme is said to be *oblivious* if the choice of cells probed is determined solely by the input set S, and does not depend on the values of the probed cells. Formally, an *oblivious query scheme* for a given representation is a set

$$Q = \{ q_i^S(u_1, \ldots, u_s) \colon i \in [n],\ S \subseteq [n] \}$$

of Boolean functions of Boolean s-vectors. For each graph $G \in \mathscr{G}$ and each query S, the value of $q_i^S(f_1(G), \ldots, f_s(G))$ must be 1 iff the output node y_i is adjacent to some input node in S. In general, the functions q_i^S should not depend on the values of all cells; for a fixed query S, we say a cell f is *probed* by Q if q_i^S depends on f for some $1 \leqslant i \leqslant n$. The time t for a given query S is defined to be the number of cells probed by Q for that query. This kind of scheme is called "oblivious" in contrast to schemes in which cells are probed one by one and their values may influence the choice of the next cell to probe; such nonoblivious schemes may be defined formally by associating a decision tree with each input set S. In either case, we are interested in the worst-case time over all queries S and all graphs $G \in \mathscr{G}$.

4. Our results

Clearly, there is a tradeoff between the space for a representation and the (worst-case) time for an associated query scheme. At least n^2 size is required, since there are 2^{n^2} graphs to be represented. The trivial representation discussed above meets this size lower bound and achieves time n^2. Since there are n bits of output, at least n time is required. A scheme with n cells for each possible input set S achieves this time bound, but at the expense of $n2^n$ size. We wish to determine the shape of the time-size tradeoff curve between these two extreme points. In particular, what time can be achieved if size must be polynomial?

The above naive representations share three properties:

monotonicity: each cell is a monotone function of edge presence;

separability: each cell is a function only of edges entering a single output node;

simplicity: each cell is an *or* of edge presence.

Indeed, these seem natural properties for representations. Note that simplicity implies monotonicity.

We call a query scheme *monotone* (separable, simple) if its associated representation is monotone (separable, simple).

Our results are as follows.

(1) For oblivious monotone schemes, and for separable schemes, $t = \theta(n^2/\log s)$ is optimal, and can be achieved using a simple, oblivious scheme.

(2) For nonoblivious, nonseparable simple schemes, $t = \theta(n^2/\log^2 s)$ is optimal.

For any positive constant ϵ, the upper bound in result (1) holds for $n^{2+\epsilon} \leqslant s \leqslant n2^n$, and the upper bound in result (2) holds for $n^{2+\epsilon} \leqslant s \leqslant 2^{n^{1/3}}$.

Two implications are:

- for oblivious schemes, monotonicity and separability are essentially equivalent restrictions;
- for separable schemes, there is a *simple* oblivious scheme that is as good as any nonoblivious scheme.

The first thing to notice is that the lower bounds support our belief that preprocessing can help very little if size is restricted to be polynomial. For the restricted classes of schemes for which we have results, preprocessing reduces the time by at most a polylogarithmic factor.

Second, it is interesting (though not surprising) that nonobliviousness helps, at least for monotone schemes. It is not clear by how much it helps; our nonoblivious lower bound only holds for simple schemes. One might hope to extend this lower bound to hold for monotone schemes. However, using slice functions it is easy to show that given a nonoblivous nonmonotone scheme with size s and time t, one can obtain a nonoblivious monotone scheme with size $n^2(s+1)$ and time $t + 2\log n$. Consequently, a good lower bound for nonoblivious monotone schemes would yield a good lower bound for nonoblivious nonmonotone schemes.

We leave as an open problem the characterization of the time-space tradeoff curve for unrestricted schemes.

In the remainder of this paper, we prove the results (1) and (2). Our proof for result (1) proceeds as follows: we first prove that from an oblivious monotone scheme, one can obtain an oblivious separable scheme. Second, we prove a lower bound for separable schemes. Third, we give an oblivious separable simple scheme that meets the lower bound. To prove result (2), we first give a simple, nonoblivious scheme that achieves the stated bound; then we prove a lower bound for simple schemes.

4.1. An oblivious monotone scheme might as well be separable

In this section, we prove that an oblivious monotone scheme might as well be separable. [1]

We define n subsets $\mathscr{G}_1, \ldots, \mathscr{G}_n$ of $\mathscr{G}$. The subset $\mathscr{G}_k$ consists of the graphs in $\mathscr{G}$ such that the output node y_j is connected to every input node if $j < k$ and to no input node if $j > k$. Every graph $G \in \mathscr{G}$ corresponds to a graph in $\mathscr{G}_k$, namely the graph obtained from G by adding all edges (x_i, y_j) where $j < k$ and removing all edges (x_i, y_j) where $j > k$. Let $G^{(k)}$ denote this graph. The graphs $G^{(k)}$ and G agree on which edges enters output node y_k.

For $k = 1, \ldots, n$, let G_k be the graph in $\mathscr{G}_k$ such that the output node y_j is connected to every input node if $j \leqslant k$ and to no input node if $j > k$.

Fix a monotone representation $F = \langle f_1, \ldots, f_s \rangle$ and a corresponding query scheme $Q = \{q_i^S\colon i \in [n],\ S \subseteq [n]\}$. To each cell $f \in F$ we assign an element $\alpha(f)$ of $[n]$ by

$$\alpha(f) = \min\{k \in [n]\colon f(G_k) = 1\}. \tag{1}$$

Since G_n is the graph with all edges present and f is a monotone function of edge presence, if $f(G_n) \neq 1$, then f is identically 0. Such cells are clearly useless, and we may assume they do not occur in the representation scheme. Therefore, we assume that every cell f is assigned a number $\alpha(f)$ by (1).

Lemma 4.1. *For any cell* f,

- f *is identically* 0 *on graphs in* $\mathscr{G}_k$ *if* $k < \alpha(f)$;
- f *is identically* 1 *on graphs in* $\mathscr{G}_k$ *if* $k > \alpha(f)$.

[1] Independently, Noam Nissan found a related proof of a result of Galbiati and Fischer [6] that monotone circuits for computing two functions from the same inputs need not share gates.

Proof. Suppose $k < \alpha(f)$. By definition of $\alpha(f)$, $f(G_k) = 0$. But each graph $G \in \mathscr{G}_k$ can be obtained from G_k by removing some edges entering output node y_k. Therefore, by monotonicity, $f(G) = 0$.

Suppose $k > \alpha(f)$. By definition of $\alpha(f)$, $f(G_{k-1}) = 1$. But each graph $G \in \mathscr{G}_k$ can be obtained from G_{k-1} by adding some edges entering output node y_k. Therefore, by monotonicity, $f(G) = 1$. □

We now construct a separable representation $\hat{F}$ by modifying the cells of F. The query scheme associated with $\hat{F}$ probes the same cells as the query scheme associated with F. The idea is as follows: in determining whether y_k is in the output set, we might as well consider $G^{(k)}$ instead of G, because these graphs agree on the edges entering output node y_k. But $G^{(k)}$ is a graph in $\mathscr{G}_k$; for such graphs, by Lemma 4.1, the cells whose α-numbers are not k give us no information about the edges entering y_k. The new query scheme determines whether y_k is in the output set from only those cells whose α-number is k.

More formally, we construct the new representation $\hat{F} = \langle \hat{f}_1, \ldots, \hat{f}_s \rangle$ as follows: Let f_k be a cell of F, and let $\alpha = \alpha(f_k)$. The corresponding cell $\hat{f}_k$ is a *projection* of f_k, obtained by substituting 1 for all edges (x_i, y_j) with $j < \alpha$ and substituting 0 for all edges (x_i, y_j) with $j > \alpha$. The new representation $\hat{F}$ is separable, and has the following properties:

(1) $\hat{f}_k$ agrees with f_k on graphs in $\mathscr{G}_\alpha$.

(2) On graphs in $\mathscr{G}_j$ where $j \neq \alpha$, both f_k and $\hat{f}_k$ are either identically 0 or identically 1, depending on whether $j < \alpha$ or $j > \alpha$.

(3) Hence f_k and $\hat{f}_k$ agree on all graphs in $\bigcup_i \mathscr{G}_i$.

We construct a query scheme $\hat{Q} = \{\hat{q}_j^S(u_1, \ldots, u_s)\}$ for $\hat{F}$ as follows: each function $\hat{q}_j^S(u_1, \ldots, u_s)$ is a projection of $q_j^S(u_1, \ldots, u_s)$ obtained by substituting 1 for each u_k where $\alpha(f_k) < j$ and substituting 0 for each u_k where $\alpha(f_k) > j$. Thus $\hat{q}_j^S$ depends only on cells $\hat{f}_k$ such that $\alpha(f_k) = j$.

We need only check that the new query scheme $\hat{Q}$ works correctly for each graph G, each query S, and each output node y_j. We assume that Q works, so the value of

$$q_j^S(f_1(G), \ldots, f_s(G)) \tag{2}$$

is correct, i.e., its value is 1 iff y_j is in the output set. Since G and $G^{(j)}$ agree on edges entering y_j, this value of (2) must equal that of

$$q_j^S(f_1(G^{(j)}), \ldots, f_s(G^{(j)})). \tag{3}$$

The value of a cell f_i on $G^{(j)}$ is guaranteed to be 1 if $\alpha(f_i) < j$ and 0 if $\alpha(f_i) > j$, so by construction of $\hat{q}_j^S$, the value of (2) must equal

$$\hat{q}_j^S(f_1(G^{(j)}), \ldots, f_s(G^{(j)})). \tag{4}$$

Since f_i and $\hat{f}_i$ agree on graphs in $\mathscr{G}_j$, the value of (4) must equal

$$\hat{q}_j^S(\hat{f}_1(G^{(j)}), \ldots, \hat{f}_s(G^{(j)})). \tag{5}$$

Finally, since $G^{(j)}$ and G agree on edges entering y_j, and since $\hat{q}_j^S$ depends only on cells that are functions of the edges entering y_j, the value of (5) must equal

$$\hat{q}_j^S(\hat{f}_1(G), \ldots, \hat{f}_s(G)). \tag{6}$$

This shows that the new query scheme $\hat{Q}$ works correctly, and completes the proof that an oblivious monotone scheme might as well be separable.

4.2. A lower bound for separable schemes

We next prove a lower bound of $t = \Omega(n^2/\log s)$ for separable schemes, using a fooling-set argument. The proof does not depend on monotonicity or even obliviousness. A separable scheme can be divided into n independent schemes, each for a graph with n input nodes but only one output node. It is therefore sufficient to prove a lower bound of $t = \Omega(n/\log s)$ for such a one-output-node scheme. We adapt the notation to one-output-node schemes in a natural way. Assume that n is even. To prove the lower bound, we consider only queries S such that $|S| = n/2$. For each such query S, let $G(S)$ be the graph (with n input nodes $x_1, \ldots, x_n$ and one output node y) containing the edges $\{(x_i, y): i \notin S\}$. If the query is S and the graph is $G(S)$, then y is not in the output "set"—no edge in $G(S)$ leads from an input node in the query to y.

Let S_1 and S_2 be distinct queries of size $n/2$. Suppose query S_1 with graph $G(S_1)$ probes the same set of cells and sees the same value for each probed cell as query S_2 with graph $G(S_2)$. Then query S_1 with graph $G(S_2)$ would also probe these cells and see the same values, and would therefore give the same answer as S_1 with graph $G(S_1)$. For query S_1 with graph $G(S_2)$, node y is in the output set, whereas for query S_1 with graph $G(S_1)$, node y is not in the output set. One of the answers must be wrong.

We have shown that the information gathered in answering any query S of size $n/2$ must be sufficient to distinguish the query from every other query of size $n/2$. The number of such queries is $\binom{n}{n/2}$. The number of ways of choosing t cells out of the s possible cells, and assigning Boolean values to each, is $\binom{s}{t}2^t$. It follows that

$$\binom{s}{t}2^t \geqslant \binom{n}{n/2}. \tag{7}$$

Using the fact that $\binom{n}{n/2} = \Omega(2^n/\sqrt{n})$, we obtain the desired lower bound $t = \Omega(n/\log s)$.

4.3. A matching upper bound for separable schemes

We next describe a simple, separable, oblivious scheme that shows that the lower bound of Section 4.2 is tight for $n^{2+\epsilon} \leqslant s \leqslant n2^n$. It is sufficient to describe a one-output-node scheme that achieves time $t = O(n/\log s)$ and space s/n; constructing one copy of this scheme for each of the n output nodes yields a scheme achieving space s and $O(n^2/\log s)$ time.

For any $n^2 \leqslant s \leqslant n2^n$, let $p = \lfloor \log(s/n^2 + 1) \rfloor$. Partition the set of input nodes into n/p groups $A_1, \ldots, A_{n/p}$ of size p. We define the representation scheme as follows: For each group A_i and each nonempty subset $X \subseteq A_i$, there is a cell f_X. For a one-output-node graph G, the value of $f_X(G)$ is 1 if G contains an edge (x_i, y) such that $x_i \in X$. This representation scheme is simple and has size $(n/p)(2^p - 1) \leqslant s/n$. To determine whether y is in the output set for a given query S, we let $X_i = S \cap A_i$ for $i = 1, \ldots, n/p$. We output the value of $\bigvee_{i=1}^{n/p} f_{X_i}(G)$. The number of cells probed is n/p, which is $O(n/\log s)$ for $s \geqslant n^{2+\epsilon}$.

```
Q1  Let S be the query.
Q2  To initialize, let T := ∅.
Q3  For i := 1 to n/p,
Q4      For j = 1,..., n/p,
Q5          probe the cell f_{S∩A_i, B_j−T}.
Q6          If the cell's value is 1,
Q7              For each y_k ∈ B_j − T,
Q8                  probe f_{S∩A_i,{y_k}}
Q9                  If the cell's value is 1, add y_k to T.
```

Fig. 1. A nonoblivious procedure for handling queries.

4.4. A simple, nonseparable, nonoblivious scheme

Next we describe a simple, nonseparable scheme that does better than separable schemes. We shall show that for $n^{2+\epsilon} \leqslant s \leqslant 2^{n^{1/3}}$, there is a scheme that achieves $t = O(n^2/\log^2 s)$, beating the lower bound of Section 4.2. Let $p = \log(\sqrt{s}/n)$. We partition the input nodes into n/p groups $A_1, \ldots, A_{n/p}$ of size p, and also partition the output nodes into n/p groups $B_1, \ldots, B_{n/p}$ of size p. For each group A_i, each subset X of A_i, each group B_j, and each subset Y of B_j, we have a cell $f_{X,Y}$ that is 1 iff there is some edge from an input node in X to an output node in Y. The space required for this representation scheme is $((n/p)2^p)^2 \leqslant s$.

To describe the nonoblivious query scheme for this representation, we use the procedure in Fig. 1.

The total number of executions of step Q5 is $(n/p)^2$. If $f_{S \cap A_i, B_j - T}$ is 1, there must be some output node $y_k \in B_j - T$ that is connected to an input node in S. Therefore, at least one output node is added to T during the execution of the loop consisting of steps Q7 through Q9. This loop has at most $|B_j| = p$ iterations, and at most n nodes are added to T during the entire process, so the total number of executions of step Q8 is np. The total number of cells probed during the execution of the procedure of Fig. 1 is thus $(n/p)^2 + np$, which is $O(n^2/\log^2 s)$ when $n^{2+\epsilon} \leqslant s \leqslant 2^{n^{1/3}}$.

4.5. A lower bound for simple schemes

In this section, we show that no simple scheme is asymptotically better than that of Section 4.4. Fix a simple representation scheme F consisting of s cells, and an associated query scheme Q (not necessarily oblivious). Each cell f of F is the *or* of a collection $E(f)$ of possible edges of G. Let $in(f)$ be the set of input nodes with incident edges in $E(f)$, and let $out(f)$ be the set of output nodes with incident edges in $E(f)$. We say f is *tall* if $|in(f)| \geqslant 2 \log s$, and is *wide* if $|out(f)| \geqslant 2 \log s$. We say f is *small* if f is neither tall nor wide. If f is small, then $|E(f)| < 4 \log^2 s$. Assume that n is even.

Claim. *There is an $n/2$-element subset A of the input nodes such that for every tall f in F, $in(f) \cap A$ is nonempty.*

Proof. We use the probabilistic method. The number of ways of choosing an $n/2$-element subset A of the input nodes is $\binom{n}{n/2}$. For any tall f, the number of ways of choosing A such that $in(f) \cap S = \emptyset$ is at most

$$\binom{n-2\log s}{n/2}.$$

Hence the probability that a randomly chosen A fails to intersect $in(f)$ is at most

$$\frac{\binom{n-2\log s}{n/2}}{\binom{n}{n/2}} = \frac{\left(\frac{n}{2}\right)\cdots\left(\frac{n}{2}-2\log s+1\right)}{(n)\cdots(n-2\log s+1)} \leqslant 2^{-2\log s} = s^{-2}.$$

Since there are at most s tall f's, the probability that there exists even one f such that $in(f) \cap A = \emptyset$ is at most $1/s$. □

By the same argument, there exists an $n/2$-element subset B of the output nodes intersecting every wide f in F. Let G be the graph with edges $\{(x_i, y_j): x_i \in A \text{ or } y_i \in B\}$. For every tall or wide cell f, among the edges $E(f)$ of which f is the *or*, at least one is present. Thus every tall cell and every wide cell has value 1 for G. Let $\bar{A}$ be the set of input nodes not in A, and let $\bar{B}$ be the set of output nodes not in B.

Lemma 4.2. *For every possible edge $(x_i, y_j) \in \bar{A} \times \bar{B}$, some small cell $f \in F'$ that depends on (x_i, y_j) must be probed for the query $S = \bar{A}$.*

Proof. For the query $S = \bar{A}$, the output set is $T = B$. Consider a possible edge $(x_i, y_j) \in \bar{A} \times \bar{B}$, and let G' be the graph obtained from G by adding the edge (x_i, y_j). For the same query $S = \bar{A}$, the output set T' is now $B \cup \{y_j\}$. Since the appropriate output has changed, the correctness of the query scheme demands that the value of one of the probed cells must change. But each of the tall and wide cells still has value 1 for G', so one of the small probed cells must depend on the edge (x_i, y_j). □

Each small cell depends on fewer than $4 \log^2 s$ possible edges, and there are $n^2/4$ possible edges in $\bar{A} \times \bar{B}$. It follows that $\Omega(n^2/\log^2 s)$ cells are probed for query $S = \bar{A}$. This completes the proof of the lower bound for simple query schemes.

Acknowledgment

Thanks to Mike Saks and Ron Rivest for helpful comments.

References

[1] A. Borodin, L. Guibas, N. Lynch and A. Yao, Efficient searching using partial ordering, *Inform. Process. Lett.* **12** (1981) 71–75.
[2] M.L. Fredman, Lower bounds on the complexity of some optimal data structures, *SIAM J. Comput.* **10** (1981) 1–10.
[3] M.L. Fredman, A lower bound on the complexity of orthogonal range queries, *J. ACM* **28** (1981) 696–705.
[4] M.L. Fredman, The complexity of maintaining an array and computing its partial sums, *J. ACM* **29** (1982) 25–260.

[5] M.L. Fredman and D.J. Volper, Query time versus redundancy trade-offs for range queries, *J. Comput. System Sci.* **23** (1981) 28–34.
[6] G. Galbiati and M. Fischer, On the complexity of 2-output boolean networks, *Theoret. Comput. Sci.* **16** (1981) 177–185.
[7] A.C. Yao, Should tables be sorted?, *J. ACM* **28** (1981) 615–628.
[8] A.C. Yao, On the complexity of maintaining partial sums, *SIAM J. Comput.* **14** (1985) 277–288.
[9] A.C. Yao, On the space-time tradeoff for answering range queries, to appear.