SANDIA REPORT

SAND94–3128 • UC–705
Unlimited Release
Printed May 1995

On Certificates and Lookahead in Dynamic Graph Problems

Sanjeev Khanna, Rajeev Motwani, Randall H. Wilson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

SF2900Q(8-81)

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND94–3128
Unlimited Release
Printed May 1995

Distribution
Category UC–705

On Certificates and Lookahead in Dynamic Graph Problems

Sanjeev Khanna* Rajeev Motwani†
Department of Computer Science
Stanford University
Stanford, CA 94305

Randall H. Wilson‡
Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-0951

May 5, 1995

Abstract

Recent work in dynamic graph algorithms has led to efficient algorithms for dynamic *undirected* graph problems such as connectivity. However, no efficient algorithms are known for the dynamic versions of fundamental *directed* graph problems like strong connectivity and transitive closure, as well as some undirected graph problems such as maximum matchings and cuts. We provide some explanation for this lack of success by presenting quadratic lower bounds on the *certificate complexity* of the seemingly difficult problems, in contrast to the known *linear* certificate complexity for the problems which have efficient dynamic algorithms. A direct outcome of our lower bounds is the demonstration that a generic technique for designing efficient dynamic graph algorithms, viz., sparsification, will not apply to the difficult problems. More generally, it is our belief that the boundary between tractable and intractable dynamic graph problems can be demarcated in terms of certificate complexity.

In many applications of dynamic (di)graph problems, a certain form of *lookahead* is available. Specifically, we consider the problems of assembly planning in robotics and the maintenance of relations in databases. These give rise to dynamic strong connectivity and dynamic transitive closure problems, respectively. We explain why it is reasonable, and indeed natural and desirable, to assume that lookahead is available in these two applications. Exploiting lookahead to circumvent their inherent complexity, we obtain efficient fully-dynamic algorithms for strong connectivity and transitive closure.

*Supported by an OTL grant, and NSF Grant CCR-9357849.

†Supported by Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, an OTL grant, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡Supported by the Stanford Integrated Manufacturing Association, and by Sandia National Laboratories under DOE contract DE-AC04-94AL85000. This work was performed in part while the author was at the Department of Computer Science, Stanford University.

1 Introduction

A dynamic graph problem is that of efficiently maintaining some property of a graph undergoing structural changes defined by update operations such as insertion and deletion of edges, so as to minimize the amount of recomputation needed after each update. An "efficient" algorithm for such problems will be able to incrementally maintain the graph property at a cost (per update operation) that is significantly smaller than the cost of computing the graph property from scratch.

While much of the prior research in the design of dynamic graph algorithms involved problem-specific approaches [9, 10, 11, 19, 24], recently Eppstein, Galil, Italiano, and Nissenzweig [8] presented a general paradigm called *sparsification* that is useful in both designing new dynamic graph algorithms, as well as speeding up the existing ones. The sparsification technique is based on the notion of a *sparse strong certificate* which, informally speaking, is a sparse graph whose structure is representative of the input graph with respect to the property of interest. Eppstein *et al* established the existence of sparse strong certificates for a variety of *undirected* graph properties[§] such as edge and vertex connectivity, bipartiteness, and minimum spanning tree, thereby obtaining efficient dynamic algorithms; recently, Henzinger and King [14] improved some of these results and obtained randomized fully-dynamic algorithms requiring only *poly-logarithmic* time per operation.

In practice, two particularly important dynamic graph problems are *strong connectivity* (in industrial robotics applications [25]) and *transitive closure* (in database applications [28]). Despite intensive efforts, no efficient dynamic graph algorithms have been devised for these two problems, as well as a host of others. It appears that known techniques are of little use for non-trivial *directed* graph problems. We attempt to explain this lack of success by initiating a study of the *strong certificate complexity* of graph properties. For a variety of graph properties, we establish the existence of n-vertex graphs for which every strong certificate must have $\Omega(n^2)$ edges, thereby ruling out the application of sparsification. We also show that similar bounds hold even when a more relaxed notion of certificates is used. Our results demonstrate a qualitative distinction between the certificate complexity of fully-dynamic graph problems with known efficient algorithms and the (primarily digraph) problems for which no efficient algorithms are known. This provides a partial explanation for the lack of progress on some dynamic graph problems.

Fortunately, many real systems have some form of *lookahead* available, i.e., the dynamic algorithm is provided with information about future updates. In the Appendix, we detail a problem in industrial robotics —*assembly planning* [15, 25, 26, 27] — that requires an efficient dynamic algorithm for strong connectivity, and indicate why lookahead is naturally available in this setting. Also, the problem of dynamic transitive closure arises in numerous database applications (see Yannakakis [28]). This is equivalent to maintaining the transitive closure of a relation undergoing frequent updates, and it is usually desirable to update the transitive closure only after accumulating batches of updates, leading to the availability of lookahead. While the issue of lookahead is receiving increasing attention in *online* algorithms [3, 17, 18], there has not been much work along these lines for dynamic graph algorithms. Schaeffer and Varman [20] studied parallel batch update of minimum spanning trees, while Eppstein [7] considered the offline version of the same problem. Dai, Imai, Iwano, and Katoh [6] describe an approach for constructing semi-online algorithms, i.e., algorithms based on partial knowledge of future deletes, using partially dynamic algorithms, and apply their techniques to *undirected* graph connectivity problem.

We propose the study of lookahead as a means to circumvent the inherent complexity of certain dynamic

[§]As we explain in Section 2, the word *property* is used in a broad sense to include non-boolean functions on graphs.

graph problems. We present a lookahead-based variant of sparsification and use it to obtain efficient fully-dynamic algorithms for the digraph properties that motivated our investigation, viz., strong connectivity and transitive closure. Our algorithms require only a *weak form of lookahead* where it suffices to know only the vertices involved rather than the exact sequence of edge operations.

In Section 2, we briefly review sparsification in a slightly more general framework than that of Eppstein *et al* [8]. Section 3 develops the concept of *strong certificate complexity* and *witness complexity*. We present a theorem relating these two measures and hence obtain a general technique for bounding the certificate complexity. We present quadratic lower bounds on this complexity for a variety of problems. We also show that these bounds essentially continue to hold under a relaxed notion of certificates. We conclude this section by pointing out some intriguing trends that emerge from our results, and pose a couple of conjectures. In Section 4, we present a framework for incorporating lookahead into dynamic graph algorithms via a variant of sparsification. Section 5 is devoted to the presentation of two algorithms for strong connectivity using our new framework; the Appendix contains a description of the motivating application, assembly planning in robotics, including a technical development of this application and the practical implications of our results. Finally, in Section 6, we extend our results to the transitive closure problem with lookahead and provide a brief discussion of the relevance of this problem in database applications such as maintenance of materialized views.

2 Preliminaries

We view a graph G as a set of edges whose cardinality is denoted $|G|$, enabling us to employ set operators to graphs. The underlying vertex set will be a fixed set of n labeled vertices, or will be clear from the context.

The sparsification technique was originally presented in a framework for boolean graph properties, along with applications to non-boolean functions such as minimum spanning tree [8]. We review the sparsification technique using a more general formalism directly applicable to non-boolean graph properties.

Let $\mathcal{P}$ denote an arbitrary function, also referred to as a *graph property*, which maps graphs to *non-empty sets of objects*. For example, the minimum spanning forest function will presumably map a given weighted undirected graph to several possible minimum weight spanning forests. In case of boolean properties, $\mathcal{P}$ may map graphs to symbols TRUE and FALSE. The sparsification technique is based on the notion of certificates.

Definition 1 (Strong Certificate) *For any graph property $\mathcal{P}$, a* strong $\mathcal{P}$-certificate *of a graph G is a graph G' on the same vertex set as G such that for any graph H, $\mathcal{P}(G' \cup H) \subseteq \mathcal{P}(G \cup H)$.*

Definition 2 (Sparse Strong Certificate) *A property $\mathcal{P}$ has* sparse strong certificates *if for every graph G, there exists a strong $\mathcal{P}$-certificate for G with $O(n)$ edges.*

We claim that the next two propositions due to Eppstein *et al* [8] hold under the extended definition too.

Proposition 1 (Transitivity) *If G' is a strong $\mathcal{P}$-certificate for G, and G'' is a strong $\mathcal{P}$-certificate for G', then G'' is a strong $\mathcal{P}$-certificate for G.*

Proposition 2 (Monotonicity) *If G' is a strong $\mathcal{P}$-certificate for G, and H' is a strong $\mathcal{P}$-certificate for H, then $G' \cup H'$ is a strong $\mathcal{P}$-certificate for $G \cup H$.*

Now suppose $\mathcal{P}$ is a property with sparse strong certificates. Propositions 1 and 2 together imply that a recursive divide-and-conquer approach can be used for computing the sparse strong certificates. The basic idea in the sparsification technique is to maintain a complete binary tree with $\lceil |G|/n \rceil$ leaves such that each leaf represents n edges of the graph. Each internal node maintains a sparse strong certificate of the graph formed by the union of its two children. Using the transitivity and the monotonicity properties, it follows that the root contains a sparse strong certificate for the whole graph. As the edges are inserted and deleted from the graph, the certificates along the appropriate leaf-root paths are updated. Since each certificate is sparse, an update on the whole graph now reduces to $O(\log(|G|/n))$ updates on sparse graphs. In essence, this technique has cost which is the same as that incurred while processing updates on a sparse graph.

3 Lower Bounds on Strong Certificate Size

The sparsification technique allows us to devise efficient dynamic algorithms for a host of graph properties including connectivity and minimum spanning trees. However, sparsification has its limitations: many dynamic graph properties such as strong connectivity, maximum matchings, transitive closure and graph search trees, do not yield to sparsification. In this section we explain this phenomenon by establishing the lack of sparse strong certificates for these properties; in particular, we establish that they have instances requiring quadratic-sized certificates, even for a very *relaxed notion of certificates.*

Definition 3 (Property Values) *A graph property $\mathcal{P}$ is said to be* mono-valued *if for any graph G, $\mathcal{P}(G)$ is a singleton; otherwise, it is said to be a* multi-valued property.

For example, all boolean graph properties are mono-valued. However, the class of mono-valued graph properties is a strictly larger class as it also includes properties such as transitive closure.

Definition 4 (Critical Graph) *A graph G is called* $\mathcal{P}$-critical *if for every proper subgraph G' of G, we have $\mathcal{P}(G') \neq \mathcal{P}(G)$.*

Definition 5 (Witness Complexity) *Let $\mathcal{H}$ denote the family of all n-vertex $\mathcal{P}$-critical graphs for a mono-valued property $\mathcal{P}$. Then the* witness complexity *$g(n)$ of $\mathcal{P}$ is defined as* $\max_{G \in \mathcal{H}} |G|$.

Definition 6 (Distinguishing Graph) *Given two graphs G_1 and G_2, a graph H is called a* $\mathcal{P}$-distinguishing graph *for G_1 and G_2 if $\mathcal{P}(G_1 \cup H) \neq \mathcal{P}(G_2 \cup H)$.*

Let $\mathcal{C}(G)$ denote the set of graphs G' which are strong $\mathcal{P}$-certificates for G. A graph $G' \in \mathcal{C}(G)$ if and only if there does not exist a $\mathcal{P}$-distinguishing graph for G and G'.

Definition 7 (Strong Certificate Complexity) *Let $\mathcal{G}$ denote the family of all n-vertex graphs. Then the* strong certificate complexity *$f(n)$ of a graph property $\mathcal{P}$ is defined as*

$$\max_{G \in \mathcal{G}} \min_{G' \in \mathcal{C}(G)} |G'|.$$

3.1 Mono-Valued Properties and Lower Bounds via Witness Complexity

We show that the witness complexity of a mono-valued property $\mathcal{P}$ can be used to derive lower bounds on its strong certificate complexity.

Theorem 1 *Let $\mathcal{P}$ be a mono-valued graph property with witness complexity $\Omega(g(n))$. Then the strong certificate complexity of $\mathcal{P}$ is $\Omega(g(n)/\log n)$.*

The proof proceeds as follows. Let G be a largest $\mathcal{P}$-critical graph and let $\mathcal{F}$ denote the family of all labeled subgraphs of G. We claim that for any distinct $G_1, G_2 \in \mathcal{F}$, $\mathcal{C}(G_1) \cap \mathcal{C}(G_2) = \emptyset$. Suppose not; let $G' \in \mathcal{C}(G_1) \cap \mathcal{C}(G_2)$. Assume without loss of generality that $|G_1| \geq |G_2|$, and let $H = G \setminus G_1$. Clearly, $G_1 \cup H = G$. On the other hand, by assumption, G_1 must contain at least one edge not in G_2. Therefore, $(G_2 \cup H) \subset G$ and, since G is $\mathcal{P}$-critical, we conclude that $\mathcal{P}(G_1 \cup H) \neq \mathcal{P}(G_2 \cup H)$, thereby contradicting the existence of such a G'. Thus, if $f(n)$ denotes the strong certificate complexity of $\mathcal{P}$, we must have

$$\sum_{i=0}^{f(n)} \binom{n(n-1)}{i} \geq |\mathcal{F}|.$$

Using Stirling's approximation and the fact that $|\mathcal{F}| = 2^{\Omega(g(n))}$, we obtain the desired result.

Corollary 1 *The following (di)graph properties have $\Omega(n^2/\log n)$ strong certificate complexity: (1) transitive closure; (2) diameter; and, (3) minimum cut value.*

Each of these three properties is mono-valued and we obtain the results by applying Theorem 1 to the following critical graphs: (1) a directed bipartite graph with all edges directed from one bipartition to the other; (2) the complete bipartite graph $K_{n,n}$ (diameter two); and, (3) the complete graph K_n. With some further work, all three lower bounds may be improved to $\Omega(n^2)$. The details and some other interesting applications of this theorem are deferred to the final version of this paper.

3.2 Multi-Valued Properties and Properties with Sparse Witnesses

Theorem 1 has its limitations in yielding good lower bounds. Several graph properties have linear witness complexity and yet no sparse strong certificates. The properties of strong connectivity and perfect matching are two such examples. In this section, we study several such properties and establish an $\Omega(n^2)$ lower bound on the strong certificate complexity of these properties.

We first show that the certificate complexity of strong connectivity is $\Omega(n^2)$.

Lemma 1 *Let G_1, G_2 be two labeled directed graphs on the same set of vertices, say V, and let H_1 and H_2 respectively denote their transitive closure graphs. Then if $H_1 \not\subseteq H_2$, there exists a graph H such that $G_1 \cup H$ is strongly connected and $G_2 \cup H$ is not.*

The proof proceeds as follows. Consider an edge $e = (x, y)$ such that $e \in H_1$ and $e \notin H_2$. Let H be the set of edges (y, z) and (z, x) where $z \in V \setminus \{x, y\}$. Clearly, $H \cup \{e\}$, and hence $H \cup H_1$, is strongly connected. On the other hand, the edges in H cannot contribute to a path from x to y and hence there is no such path in $H \cup H_2$. The lemma now follows by observing that for any two graphs G and H, we have $TC(G \cup H) = TC(TC(G) \cup H)$.

Consider now the directed bipartite graph $G = (X \cup Y, E)$ where $|X| = |Y| = n$ and E contains an edge (x, y) for all $x \in X$ and $y \in Y$. It is easily seen that $TC(G) = G$. We claim that for any graph H such that $G \not\subseteq H$, $TC(G) \neq TC(H)$. To see this, consider such a graph H and an edge $(x, y) \in G \setminus H$. Suppose there is a path from x to y in H. Such a path in H must contain one of the following: an edge between two vertices in X, an edge between two vertices in Y, or an edge from Y to X. But none of these edges are present in $G = TC(G)$. Therefore, $TC(G) \neq TC(H)$. On the other hand, if there is no path from x to y in H, then $TC(G) \neq TC(H)$. Thus, by Lemma 1, there exists a graph which distinguishes G and H for the property of strong connectivity. We conclude that any certificate of the graph G must contain every edge in the graph G.

Theorem 2 *The strong certificate complexity of strong connectivity is* $\Omega(n^2)$.

We now turn to the problem of maintaining a maximum matching in an undirected graph. Consider the bipartite graph $G = (X \cup Y, E)$ where $|X| = 2n$, $Y = Y_1 \cup Y_2$, $|Y_1| = |Y_2| = n$, and $E = \{(x, y) | x \in X,\ y \in Y_1\}$. Observe that it is useless for a certificate G' of G to have any edge $e \notin E$ because such an edge does not belong to any maximum matching of G. Thus $G' \subseteq G$. We claim that G' must have at least $n^2 + 1$ edges of G. Suppose not; then, there exists a vertex $y \in Y_1$ such that y is connected to no more than $k \leq n$ vertices in X in the graph G'. Let X' denote the set of neighbors of y. Consider the graph $H = (X \cup Y, E_H)$, where E_H contains all edges (x, y) such that $x \in X'$ and $y \in Y_2$. Clearly, the graph $G \cup H$ has a maximum matching of size $n + k$ while no matching of $G' \cup H$ can have size more than $n + k - 1$. We obtain the following theorem.

Theorem 3 *The strong certificate complexity maximum matching is* $\Omega(n^2)$.

Similar techniques can be used to establish quadratic complexity results for determining perfect matchings, and breadth-first and depth-first search tree maintenance; the details are deferred.

Theorem 4 *The strong certificate complexity of the breadth-first search tree property from a given vertex in a directed or undirected graph is* $\Omega(n^2)$.

Theorem 5 *The strong certificate complexity of a lexicographic-first depth-first search forest property in a directed graph is* $\Omega(n^2)$.

3.3 An Extended Notion of Certificates

So far we assumed that a certificate has the same vertex set as its underlying graph. However, conceivably, by using a larger vertex set, one may reduce the overall certificate complexity, measured now in terms of both the number of vertices and edges in the certificate. We refer to this new notion of certificates as *extended strong certificates* and their complexity as *extended strong certificate complexity*. Surprisingly, the quadratic complexity results of the Sections 3.1 and 3.2 generalize to this extended notion of certificates.

Theorem 6 *Let $\mathcal{P}$ be a mono-valued graph property with witness complexity $\Omega(g(n))$. Then the extended strong certificate complexity of $\mathcal{P}$ is $\Omega(g(n)/\log n)$.*

The proof proceeds as follows. Let $\mathcal{E}_n(.)$ denote a mapping from the set of n-vertex graphs to the set of n^*-vertex graphs such that $\mathcal{E}_n(G)$ is an extended strong certificate for a n-vertex graph G. As in Theorem 1,

we consider a largest $\mathcal{P}$-critical graph G on n vertices and define $\mathcal{F}$ as the family of all its labeled subgraphs. We claim that for any distinct $G_1, G_2 \in \mathcal{F}$, $\mathcal{E}_n(G_1) \cap \mathcal{E}_n(G_2) = \emptyset$. To see this, assume $|G_1| \geq |G_2|$, and define $H = G \setminus G_1$. By definition, $\mathcal{P}(\mathcal{E}_n(G_1) \cup \mathcal{E}_n(H)) = \mathcal{P}(G_1 \cup H) = P(G)$. On the other hand, $\mathcal{P}(\mathcal{E}_n(G_2) \cup \mathcal{E}_n(H)) = \mathcal{P}(G_2 \cup H) = \mathcal{P}(G') \neq \mathcal{P}(G)$ since $G' \subset G$. Therefore, $\mathcal{E}_n(G_1)$ and $\mathcal{E}_n(G_2)$ must be distinct. Thus, if $f(n)$ denotes the extended strong certificate complexity of $\mathcal{P}$, we must have

$$\sum_{i=0}^{f(n)} \binom{f(n)(f(n)-1)}{i} \geq |\mathcal{F}|.$$

Using Stirling's approximation and the fact that $|\mathcal{F}| = 2^{\Omega(g(n))}$, we obtain the desired result.

Theorem 7 *Suppose that for a graph property $\mathcal{P}$ there exists an n-vertex graph G with m edges such that $\mathcal{C}(G) = G$. Then the extended strong certificate complexity of $\mathcal{P}$ is $\Omega(m/\log n)$.*

The proof proceeds as follows. If $\mathcal{C}(G) = G$, we show that for any two distinct subgraphs of G, say G_1 and G_2, there exists a graph H such that $\mathcal{P}(G_1 \cup H) \neq \mathcal{P}(G_2 \cup H)$. To see this, assume $|G_1| \geq |G_2|$ and define $H_1 = G \setminus G_1$. Clearly, $G_1 \cup H_1 = G$ and $G_2 \cup H_1 = G' \subset G$. But $G' \notin \mathcal{C}(G)$ and, therefore, there exists a graph H_2 such that $\mathcal{P}(G \cup H_2) \neq \mathcal{P}(G' \cup H_2)$. Now the argument used in Theorem 6 implies that $\mathcal{E}_n(G_1) \cap \mathcal{E}_n(G_2) = \emptyset$. Since there are 2^m possible subgraphs of G, the result follows from the counting argument used earlier.

Corollary 2 *The extended strong certificate complexity is $\Omega(n^2/\log n)$ for each of the following (di)graph properties: (1) transitive closure; (2) diameter; (3) minimum cut; (4) strong connectivity; (5) maximum matching; (6) breadth-first search tree; and, (7) lexicographic-first depth-first search.*

The lower bound for the first three properties are an immediate corollary of Theorem 6 while those for the last four can be obtained from Theorem 7.

3.4 Canonical Hard Instances and A Conjecture

We conclude our discussion of certificate complexity by commenting on some interesting trends that emerge from our work.

It is rather surprising that the "hard" instances for most of the properties were simply bipartite graphs. A natural question is: do bipartite graphs provide canonical hard instances for strong certificate complexity? The following theorem asserts that this is almost the case. The proof is deferred to the final version.

Theorem 8 *Let $f(n)$ denote the strong certificate complexity of a property $\mathcal{P}$, and $h(n)$ denote the strong certificate complexity of $\mathcal{P}$ when the input is restricted to bipartite graphs. Then, $h(n) = \Omega(f(n)/\log n)$.*

A rather intriguing trend which seems to emerge is that a somewhat restricted class of graph properties appears to have either linear or quadratic strong certificate complexity, but not in between. We define a *non-degenerate* graph property as a graph property $\mathcal{P}$ such that there exist two graphs $G_1, G_2 \in \mathcal{P}$ such that G_1 has $O(n)$ edges while G_2 has $\Omega(n^2)$ edges. (An example of a degenerate property would be a threshold function on the number of edges with a super-linear threshold.)

Conjecture 1 *Let $\mathcal{P}$ be a non-degenerate, monotone graph property. Then the strong certificate complexity of $\mathcal{P}$ is either $O(n)$ or $\Omega(n^2)$.*

In light of our results and the above conjecture, a review of known efficient dynamic graph algorithms suggests that a non-degenerate, monotone graph problem has an efficient dynamic algorithm only if it has sub-quadratic certificate complexity.

4 Sparsification with Lookahead

The strong certificate complexity results of the previous sections provide some explanation for the lack of efficient dynamic graph algorithms for a variety of problems. However, these problems routinely arise in many practical applications and there is a need for efficient algorithms. Fortunately, at least some of these applications have the crucial feature of a *lookahead* in terms of some knowledge about the updates to be performed in the future. This motivates us to study the complexity of maintaining some of these seemingly intractable properties under lookahead. While doing so, we assume only a weak form of lookahead: we know only the set of vertices involved in the updates to be performed in the immediate future. In other words, we are aware of the current hot spots of activity in the underlying graph.

Our results are based on a new notion of sparsification, namely, *lookahead-based sparsification*. The sparsification technique as presented by Eppstein *et al* [8] is essentially an edge-sparsification technique in which the sparse certificates encapsulate the essential structure of the original (dense) graph into a sparse set of edges. The lookahead-based sparsification, on the other hand, is a vertex-sparsification technique where we use the knowledge of the future to construct a representative graph H on a smaller set of vertices such that processing the edge operation sequence on the original graph is "equivalent" to performing the sequence on this smaller (possibly dense) graph H.

We defer the details of a formal lookahead-based sparsification framework to the final version of the paper. We develop the basic technique in the next two sections through applications to strong connectivity and transitive closure.

5 Assembly Sequencing and Exploiting Lookahead in Strong Connectivity

We describe the assembly planning problem in detail in the Appendix. For our purposes, the assembly sequencing problem can be abstracted as follows: given a graph G and a sequence σ of t edge insertions and deletions, determine for each $i \in [1 \ldots t]$ whether the graph obtained after the ith edge operation is strongly connected. The order of t could lie anywhere between n^2 and n^5 (see Appendix A). Let m denote an upper bound on the maximum number of edges the graph may have at any time. In practice, the graphs arising in this application have $\Omega(n^2)$ edges. Thus the straightforward algorithm for verifying the strong connectivity from scratch after each edge operation leads to an $\Omega(tn^2)$ solution to this problem, which could be as large as n^7 and hence impractical even for the most trivial applications.

We can improve the running time by using a lookahead of k into the sequence [¶] of edge operations. The idea is to decompose the processing into t/k phases with each phase involving precisely k edge operations. Let G_p denote the graph in the beginning of a phase p and, for $1 \leq i \leq k$, let $G_{p,i}$ denote the graph obtained after the ith edge operation within the phase p. The total number of vertices involved in the k edge insertion and deletions within a phase cannot exceed $2k$; call these vertices as the *active* vertices during the phase. We will need the following notation: A denotes the set of active vertices; E^A denotes the set of all possible

[¶] Actually, the problem involves an implicitly-defined offline sequence but, as explained in the Appendix, the huge length of the sequence makes it desirable to generate it in smaller blocks of length k so as to reduce the *space* (and time) requirement.

directed edges between the active vertices; E_p^A denotes the set of the edges actually present between the active vertices in the graph G_p; $E_{p,i}^A$ denotes the set E_p^A after the ith edge operation in the phase p; define G_p^- as the graph $G_p \setminus E^A$ and let H_p denote the induced subgraph on the set of active vertices in the transitive closure of the graph G_p^- (i.e., in the graph H_p, there is an edge from u to v if and only if G_p has a path from u to v in which u and v are the only active vertices); $H_{p,i}$ is the graph on the active vertices with precisely the edges in H_p and $E_{p,i}^A$; finally, define G_p^+ as the graph $G_p \cup E^A$.

We defer the proofs of Lemmas 2 and 3 to the final version of this paper.

Lemma 2 *If $G_{p,i}$ is strongly connected, then the graph G_p^+ is also strongly connected.*

Lemma 3 *For any two active vertices, say u and v, there exists a directed path from u to v in $G_{p,i}$ if and only if there exists a directed path from u to v in $H_{p,i}$.*

Theorem 9 *The graph $G_{p,i}$ is strongly connected iff both G_p^+ and $H_{p,i}$ are strongly connected.*

This theorem can be proved as follows. If $G_{p,i}$ is strongly connected then the graph G_p^+ must also be strongly connected by Lemma 2. By Lemma 3, the strong connectivity of $G_{p,i}$ implies that $H_{p,i}$ is strongly connected. For the converse, observe that the strong connectivity of $H_{p,i}$ implies that $\mathrm{TC}(G_{p,i})$ includes all edges in the set E^A where TC(.) denote the transitive closure of a given graph. Therefore,

$$\mathrm{TC}(G_{p,i}) \supseteq \mathrm{TC}(G_p^- \cup E^A) = \mathrm{TC}(G_p^+).$$

Thus $G_{p,i}$ must be strongly connected.

The running time is determined by how efficiently strong connectivity of the graphs G_p^+ and $H_{p,i}$ can be verified. The strong connectivity of the graph G_p^+ can be verified at the beginning of the phase in $O(m+n)$ time by simply collapsing all active vertices into a single vertex and then running the standard strong connectivity algorithm. Let $T(n,m)$ denote the time complexity of a computing transitive closure on a n-vertex graph with m edges. Then the graph H_p can be constructed in $O(T(n,m))$ time. Finally, the strong connectivity of each $H_{p,i}$ can be verified in $O(k^2)$ time. Hence the running time for each phase is given by $O(m + T(n,m) + k^3)$ and the total running time is given by $O(t(k^2 + T(n,m)/k))$. This expression is minimized when a lookahead of size $\Theta(T(n,m)^{1/3})$ is used. The running time for the whole sequence becomes $O(tT(n,m)^{2/3})$ which is an improvement over the naive approach for any $m = \Omega(T(n,m)^{2/3})$.

Observe that if the graph contains $o(T(n,m)^{2/3})$ edges, our algorithm is slower than the naive approach. This can be handled easily by adding a simple feature to the lookahead algorithm. At the beginning of each phase, the algorithm computes the number of edges in the current graph. If the graph contains $\Omega(T(n,m)^{2/3})$ edges, it uses this algorithm, else it simply uses the naive approach of running the strong connectivity algorithm for each operation in the phase.

Theorem 10 *There is an algorithm for fully-dynamic maintenance of the strong connectivity of a graph at an amortized cost of $O(\min\{m, k^2 + T(n,m)/k\})$ per operation using lookahead $k = O(T(n,m)^{1/3})$, where $T(n,m)$ denotes the time complexity of the transitive closure algorithm used.*

We now observe that the problem of determining strong connectivity of each $H_{p,i}$ within a phase p is identical to our original problem. This leads to a recursive approach with a significantly improved running time. We defer the details to the final version.

Theorem 11 *There is an algorithm for fully-dynamic maintenance of the strong connectivity of a graph at an amortized cost of $O(T(n,m)/n)$ per operation using lookahead $\Theta(n)$, where $T(n,m)$ denotes the time complexity of the transitive closure algorithm used.*

Transitive closure of a graph can be computed using boolean matrix multiplication [2]. Let $M(n)$ denote the complexity of a boolean matrix multiplication algorithm. Then for instance, if we use the best-known (but impractical) algorithm for matrix multiplication due to Coppersmith and Winograd [5] with $\mathrm{M}(n) = n^{2.376}$, then the time per operation of the recursive algorithm is $O(n^{1.376})$. On the other hand, using the more practical algorithm due to Strassen [23] with $\mathrm{M}(n) = n^{2.808}$, we achieve a bound of $O(n^{1.808})$ per operation. These should be contrasted with the naive algorithm requiring $\Omega(n^2)$ time per operation. Note that in this application, the graph has $\Omega(n^2)$ edges at most points of the update sequence.

Remark 1 *Of course, it would be desirable to avoid using matrix multiplication altogether, and it is our expectation that using a good transitive closure algorithm instead would still yield significant speed-ups in practice. However, in the next application of this technique, we show that it achieves a significant speed-up even without the use of a fast matrix multiplication algorithm.*

6 Database Updating and Exploiting Lookahead in Transitive Closure

Consider the problem of maintaining the transitive closure of a directed graph undergoing edge updates. The primary motivation for this problem is the maintenance of the transitive closure of relations in databases [28]. In several applications, it is known that the updates will be restricted to a particular portion of the database that will be referred to as the *active set*. The knowledge of an active set provides us with lookahead that can be used to significantly speed up the transitive closure computations. Specifically, we show that it is possible to maintain transitive closure at an amortized cost of $O(k^3 + n^2k/\log^2 n + mn/k)$ per operation, where k is the size of the active set (see Theorem 12 and Corollary 3). In applications where the amount of lookahead available is unrestricted, such as in the off-line construction of a persistent data structure for transitive closure, we show that it can be maintained at an amortized cost of $O(n\sqrt{mn}/\log n)$ per operation using a lookahead of $\Theta(\sqrt{m/n}\log n)$ (see Corollaries 4 and 5).

Our techniques also yield a solution to the problem of efficiently updating materialized views in databases [1, 4, 12, 16, 21, 22]. In many query-intensive database applications, materialized views, derived from a base relation at a central site, are maintained at multiple local sites. Abstractly speaking, a materialized view at a local site may simply correspond to the reachability information for a subset of vertices of the original graph. Since the base relation may be very large, there are tremendous costs associated with globally updating the views after each update. To save on such costs, a commonly employed strategy is to trade off currency for efficiency, e.g., by batching the updates and updating the views only after periodic intervals. Another possibility is to always keep the local views current with respect to the updates that affect the vertices in the local view, and to perform global updates only periodically. Our results indicate how the latter may be efficiently implemented by performing only local transitive closure computations after each update (see Theorem 13).

We sketch the extension of the techniques from Section 5 to all-pairs transitive closure computation; the ideas also apply to single-source transitive closure but we defer the details to the final version. We follow the notation from Section 5 and outline the computation done in each phase.

At the beginning of a phase p, we first compute the transitive closure of G_p^- in $O(T(n,m))$ time. For each non-active vertex v, let $\Gamma_p^{N \rightsquigarrow A}(v)$ and $\Gamma_p^{N \rightsquigarrow N}(v)$ denote, respectively, the set of all active vertices and the set of all non-active vertices that it can reach in the graph G_p^-. Similarly, for each active vertex v, let $\Gamma_p^{A \rightsquigarrow N}(v)$ denote the set of all non-active vertices that it can reach in G_p^-.

At the ith step in this phase, we compute the transitive closure of $H_{p,i}$ in $O(T(k,k^2))$ time. For each active vertex v, let $\Gamma_{p,i}^{A \rightsquigarrow A}(v)$ denote the set of all active vertices it can reach in the graph $H_{p,i}$. Now, if $\Gamma_{p,i}(v)$ denotes the set of all vertices reachable from a vertex v in $G_{p,i}$, we may compute it as follows. We begin by computing for each active vertex v,

$$\Gamma_{p,i}(v) = \Gamma_{p,i}^{A \rightsquigarrow A}(v) \cup \{ \bigcup_{x \in \Gamma_{p,i}^{A \rightsquigarrow A}(v)} \Gamma_p^{A \rightsquigarrow N}(x) \}.$$

Next, for each non-active vertex v, we compute

$$\Gamma_{p,i}(v) = \Gamma_p^{N \rightsquigarrow N}(v) \cup \{ \bigcup_{x \in \Gamma_p^{N \rightsquigarrow A}(v)} \Gamma_{p,i}(x) \}.$$

The computation of $\Gamma_{p,i}(v)$ for active vertices can be easily performed in $O(k^2 n)$ time. Similarly, a straightforward implementation has $O(kn^2)$ running time for computing $\Gamma_{p,i}(v)$ for all non-active vertices. This may be slightly improved as follows. Consider a bipartite graph $H^* = (A, V, E^*)$ such that there is an edge (x,y) in E^*, $x \in A$ and $y \in V$, if and only if $y \in \Gamma_{p,i}(x)$. It can be shown that the computation of $\Gamma_{p,i}(v)$ for a non-active vertex v reduces to answering a reachability query in the graph H^*, i.e., given $X = \{x \in \Gamma_p^{N \rightsquigarrow A}(v)\}$, find the set of vertices adjacent to X in H^*. Using the approach of Hellerstein, Klein, and Wilber [13], we can construct a data structure to perform this computation in $O(n^2 k / \log^2 n)$ time for all the non-active vertices. Thus the total running time of each phase is given by $O(T(n,m) + kT(k,k^2) + n^2 k^2 / \log^2 n)$, which is minimized when the lookahead is $k = \sqrt{T(n,m)} \log n / n$.

Theorem 12 *Given lookahead k, fully-dynamic maintenance of transitive closure can be performed at an amortized cost of $O(T(n,m)/k + T(k,k^2) + n^2 k / \log^2 n)$ per update.*

Corollary 3 *Given lookahead k, fully-dynamic maintenance of transitive closure can be performed at an amortized cost of $O(k^3 + n^2 k / \log^2 n + mn/k)$ per update.*

Corollary 4 *Fully-dynamic maintenance of transitive closure can be performed at an amortized cost of $O((n\sqrt{mn}) / \log n)$ per update using lookahead $\Theta(\sqrt{m/n} \log n)$.*

It is worthwhile to contrast these bounds with the bound of $O(nm)$ time per operation achieved by the naive algorithm which recomputes transitive closure at each step.

In the preceding algorithm, we explicitly maintained the transitive closure of the resulting graph after each update. We now describe a variant in which the update time is significantly improved by maintaining only an implicit representation of the transitive closure. This implicit representation can support reachability queries but at a significantly increased cost. The basic idea is the following: after each update during a phase, we compute only the transitive closure of $H_{p,i}$; queries are processed by using this along with the transitive closure of G_p^-. We present only the resulting time bounds and omit the details.

Theorem 13 *Suppose the updates are restricted to a set of k active vertices. Then, there exists a fully-dynamic data structure for processing reachability queries at an amortized cost of $O(n^3/k + k^3)$ per update, with a query cost that is $O(k)$ for queries involving at least one active vertex, and $O(k^2)$ otherwise.*

The preceding bounds can all be improved using fast matrix multiplication. For example, Corollary 4 can be replaced by the following.

Corollary 5 *Using fast matrix multiplication, fully-dynamic maintenance of transitive closure can be performed at an amortized cost of* $O(n^{2.188}/\log n)$ *per update using lookahead* $\Theta(n^{0.188}\log n)$.

Acknowledgments

We are grateful to Jean-Claude Latombe for sharing his insights about assembly planning and his constant encouragement. We thank Sergey Brin, Surajit Chaudhry, Waqar Hasan, Jeff Ullman, and Jennifer Widom for useful discussions about transitive closure and databases. Many thanks also to Julien Basch, Chandra Chekuri, Delphine Berthet, Philip MacKenzie, and David Strip.

References

[1] R. Agrawal and H.V. Jagdish. Materialization and Incremental Update of Path Information. In *Proceedings of Fifth International IEEE Conference on Data Engineering* (1989), pp. 374–383.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] S. Albers. The influence of lookahead in competitive on-line algorithms. Technical Report MPI-I-92-143, Max Planck Institute, Germany, 1993.

[4] J.A. Blakaley, P-A. Larson, and F.W. Tompa. Efficiently Updating Materialized Views. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (1986), pp. 61–71.

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:1–6 (1990).

[6] Y. Dai, H. Imai, K. Iwano, and N. Katoh. How to treat delete requests in semi-online problems. In *Proceedings of the 4th International Symposium on Algorithms and Computation* (1993), pp. 48-57.

[7] D. Eppstein. Offline Algorithms for Dynamic Minimum Spanning Tree Problems. *Journal of Algorithms*, 17:237-250 (1994).

[8] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – A technique for speeding up dynamic graph algorithms. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science* (1992), pp. 60–69.

[9] G.N. Frederickson. Data Structures for On-line Updating of Minimum Spanning Trees, with Applications. *SIAM Journal on Computing*, 14:781–798 (1985).

[10] G.N. Frederickson. Ambivalent Data Structures for Dynamic 2-edge Connectivity and k Smallest Spanning Trees. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science* (1991), pp. 632–641.

[11] Z. Galil and G.F. Italiano. Fully dynamic algorithms for edge-connectivity problems. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 317-327.

[12] E. Hanson. A Performance Analysis of View Materialization Strategies. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (1987), pp. 440–453.

[13] L. Hellerstein, P. Klein, and R. Wilber. On the time-space complexity of reachability queries for preprocessed graphs. *Information Processing Letters*, 35:261–267 (1990).

[14] M. Rauch Henzinger and V. King. Randomized Dynamic Algorithms with Polylogarithmic Time per Operation. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing* (1995).

[15] L.S. Homem de Mello and S. Lee (editors). *Computer-Aided Mechanical Assembly Planning*. Kluwer, 1991.

[16] H.V. Jagdish. A Compression Technique to Materialize Transitive Closure. *ACM Transactions on Database Systems*, 15:558–598 (1990).

[17] E. Koutsoupias and C.H. Papadimitriou. Beyond competitive analysis. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (1994), pp. 394–400.

[18] R. Motwani, V. Saraswat, and E. Torng. Online Scheduling with Lookahead: Multipass Assembly Lines. Report CPS-94-41, Department of Computer Science, Michigan State University, 1994.

[19] M. Rauch. Fully dynamic biconnectivity in graphs. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science* (1992), pp. 50–59.

[20] A. Schaeffer and P. Varman. Parallel Batch Update of Minimum Spanning Trees. Technical Report COMP TR90-140, Rice University, 1990.

[21] A. Segev and J. Park. Maintaining Views in Distributed Databases. In *Proceedings of Fifth International IEEE Conference on Data Engineering* (1989), pp. 262–270.

[22] O. Shmueli and A. Itai. Maintenance of Views. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (1984), pp. 240–255.

[23] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14:354–356 (1969).

[24] J. Westbrook and R.E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7:433–464 (1992).

[25] R.H. Wilson. *On Geometric Assembly Planning*. PhD thesis, Stanford University, 1992. Stanford Technical Report STAN-CS-92-1416.

[26] R.H. Wilson and J-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71 (1994).

[27] J.D. Wolter. *On the Automatic Generation of Plans for Mechanical Assembly*. PhD thesis, University of Michigan, 1988.

[28] M. Yannakakis. Graph-theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1990), pp. 230-42.

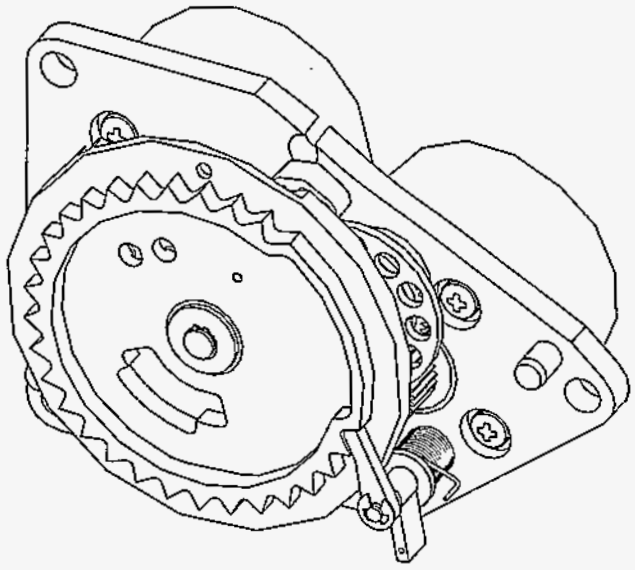

Figure 1: The discriminator assembly

A The Assembly Sequencing Problem

Automated assembly planning promises to reduce the effort required to create manufacturing plans for products, as well as provide valuable and timely design-for-assembly feedback to designers. Together with the rising use of Computer-Aided Design (CAD) systems, this opportunity has fueled a decade of research in assembly planning (see, e.g., [15]).

Assembly sequencing is a subproblem of assembly planning. Given a set of parts and their final positions in a product (for example, see Figure 1), assembly sequencing attempts to find a sequence of object motions that will build the assembly from the parts without inducing part collisions. Assuming that the parts are rigid and each operation mates two rigid subassemblies to produce a larger one, an assembly sequence can be found by disassembling the product and then reversing the operations and their order. The initial assembly is divided into two subassemblies, each of which is disassembled recursively to find the disassembly sequence.

A.1 Blocking Graphs

The *blocking graph* is a central structure in assembly sequencing, allowing polynomial-time sequencing in many situations [26, 27]. Given a rigid motion d (for instance an infinite translation along a vector $d = (\mathrm{d}x, \mathrm{d}y, \mathrm{d}z)$), a part P_1 *blocks* a part P_2 along d if and only if P_2 collides with P_1 when moved along d. The blocking graph $G(A, d)$ for an assembly A and motion d is a directed graph with one vertex for each part in A, and an arc from vertex N_i to N_j exactly when P_j blocks P_i along d. A proper subassembly S of A can be removed along d if no parts in $A \setminus S$ block parts in S. Such an S exists if and only if $G(A, d)$ is not strongly connected [25, 26].

Finally, consider the space of all possible motions d. This space can be partitioned into regions such that the blocking graph is constant for all motions in that region; furthermore, the graphs of neighboring regions differ by at most one arc. Refer to Appendix A.3 for useful classes of motion and the regions that result. To find a removable subassembly, it suffices to check all the blocking graphs for strong connectedness; this step dominates the running time of the algorithm [26]. A traversal of the regions yields a directed graph and a sequence of edge insertions and deletions to that graph. This yields the following problem: given a graph G and a sequence σ of t edge insertions and deletions, determine for each $i \in [1 \ldots t]$ if the graph obtained after the ith edge operation is strongly connected. The order of t could lie anywhere between n^2 and n^5,

depending on the type of motion used for the disassembly (see Appendix A.3).

In principle, the entire sequence σ can be computed in advance, yielding an offline version of the dynamic strong connectivity problem. However, note that the basic goal is to find a point in the sequence (i.e., a direction vector d) at which the resulting graph is *not* strongly connected. Having found this point, the graph will be decomposed into its strongly connected components, and the problem then needs to solved independently on these components. Thus, it would be extremely wasteful (in terms of both time and space) to compute the entire sequence σ in the case where the disassembly could be performed very early in the sequence, as is typically the case. Of course, we could use a doubling search technique to ensure that we only compute a sequence that is at most twice as long as necessary. However, this would require a tremendous amount of space since the sequence could be of length as much as n^7.

A more reasonable approach is to use a lookahead k significantly smaller than t, thereby ensuring that only a near-optimal prefix of the sequence σ needs to be computed, while reducing the space requirement to a quantity much smaller than the sequence length.

A.2 Practical Implications

We briefly discuss some practical aspects of the solution obtained in this paper. Figure 1 shows a *discriminator*, a safety device with 42 complex parts modeled in the ProEngineer CAD system. A version of the infinitesimal rigid motion disassembly algorithm discussed in Appendix A.3 is used in a planner that finds an assembly plan for the discriminator in about 30 seconds on an SGI Indigo^2 workstation. The planner checks several thousand blocking graphs in the planning process.

We are currently attempting to apply the assembly planner to assemblies having 1000 parts. Planning for typical industrial assemblies (such as a car) will require massive numbers of strong connectedness checks on graphs having on the order of one million edges. For such assemblies, methods such as those described in this paper will be critical, even with the use of fast matrix multiplication algorithms.

A.3 Classes of Motion

Depending on the desired output of assembly planning and the rest of the planning system in which an assembly sequencer operates, several classes of relative motions between subassemblies might be allowed. Each class generates a space of possible motions and regions in that space with the same blocking graph. We first present a more detailed view of a disassembly algorithm for a simple case, that of infinitesimal translations in two dimensions. We then summarize algorithms for two types of three-dimensional motions used in real assembly planning systems: infinite translations and infinitesimal rigid motions (see [26] for further details). Note that computing the strong connectedness of a sequence of blocking graphs dominates the algorithm's running time for each class of motion.

A Simple Case. Consider a two-dimensional polygonal assembly A such as the one shown in Figure 2. We wish to identify a subassembly that can be translated infinitesimally without colliding with any other parts, if one exists. Representing a direction of infinitesimal motion as a unit vector $d = (\mathrm{d}x, \mathrm{d}y)$, the set of all motions is the unit circle S^1. Consider one contact between an edge of a part P_i and an edge of part P_j. The diameter of S^1 parallel to the contact edge divides the set of motions into an open half-circle of motions for which P_j blocks P_i, and a closed half-circle of motions for which P_j does not block P_i at that contact. A similar division occurs if a point of P_i contacts an edge of P_j.

The set of diameters corresponding to all the contacts of the assembly divide S^1 into a set of regions. Since blocking relationships only change at the endpoints of these diameters, it is clear that the blocking graph for all points in any region is constant. Furthermore, if no two diameters coincide, the blocking graph

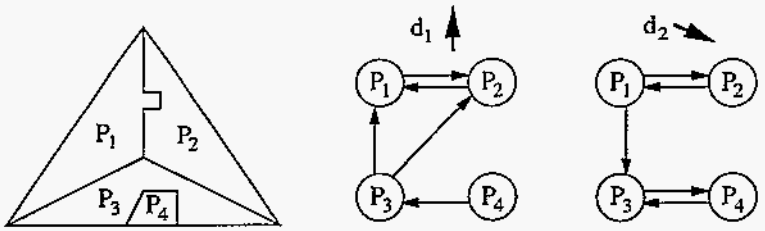

Figure 2: A simple assembly and blocking graphs for two directions of motion

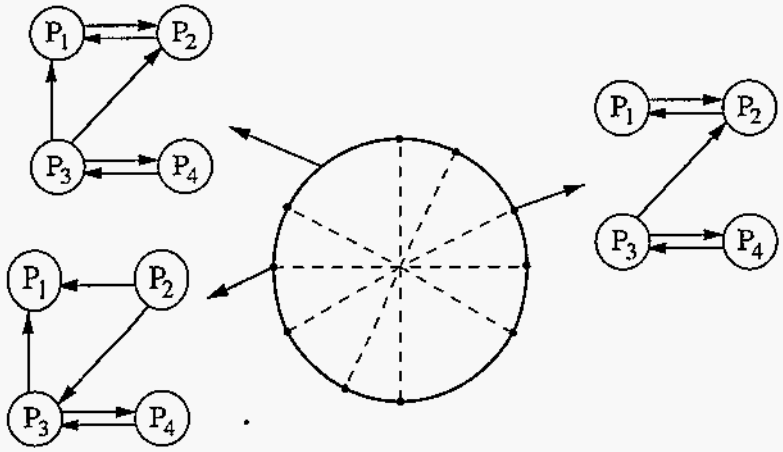

Figure 3: The arrangement on S^1 and some of the blocking graphs for the assembly in Figure 2

for a region differs by at most one arc from the blocking graph of each neighbor region. Figure 3 shows the arrangement on S^1 and some of the blocking graphs for the regions.

To identify a movable subassembly, we construct the above arrangement on S^1 and compute a blocking graph $G(A, R_0)$ for one region R_0. We then step around S^1 to each region in turn, which gives a sequence t of arc additions and deletions to the blocking graph $G(A, R_0)$. If the assembly has n parts with e edges in total, the arrangement can be constructed in $O(e \log e)$ time and has $O(e)$ regions. If the amortized time to check an n-vertex graph in the sequence for strong connectedness is $f(n)$, then identifying a movable subassembly requires $O(e(f(n) + \log e))$ time.

Infinite Translations. Consider now a more realistic case: an assembly A of polyhedra, to be disassembled using single translations to infinity. In this case, the set of translation directions can be represented by S^2. A part P_i blocks part P_j in a (generally non-convex) region of the sphere bounded by arcs of great circles; this region is computed by projecting the Minkowski difference $P_i \ominus P_j$ onto S^2 by a central projection. In an assembly with a total of v vertices, there are $O(v^2)$ boundary edges of these regions, which define an arrangement of $O(v^4)$ cells on the sphere. This arrangement can be computed in $\Theta(v^4)$ time. $O(v^4)$ blocking graphs must be checked for strong connectedness, for a total of $O(v^4 f(n))$ time.

Infinitesimal Rigid Motions. Now assume that we wish to identify subassemblies of A that can be moved an infinitesimal distance in rigid motion. Such subassemblies are a superset of all possible removable subassemblies. A direction of rigid motion can be represented as a 6D unit vector giving three degrees of translation and three degrees of rotational motion. Thus the set of motions can be represented as the surface of the unit sphere S^5. The constraints on motion imposed by contacts can be represented as finite sets of point-plane constraints [26]. Each point-plane constraint defines a 5D hyperplane through the origin, cutting S^5 along a "great sphere" (for want of a better term). The set of constraints between any two parts defines

a convex polytope on S^5; for motions within this polytope the two parts are free to move and outside of it the parts collide.

The set of convex polytopes for all contacts of A determines an arrangement on S^5, consisting of relatively-open cells of dimensions $0, \ldots, 5$. For a total of c point-plane constraints, this arrangement can have $O(c^5)$ cells, which are computable in the same time bound. In fact, the cells can be computed incrementally, along with the sequence of edge insertions and deletions to the blocking graph that they entail. Thus the running time for the local rigid motion disassembly algorithm is $O(c^5 f(n))$.

Distribution for SAND95-0722:

MS9018	Central Technical Files, 8523-2	(1)
MS 0899	Technical Library, 13414	(5)
MS 0619	Print Media, 12615	(1)
MS 0100	Document Processing for DOE/OSTI, 7613-2	(2)
MS 0951	Randy Wilson, 2121	(20)