Graph Certificates, Lookahead in Dynamic Graph Problems, and Assembly Planning in Robotics

SANJEEV KHANNA* RAJEEV MOTWANI* RANDALL H. WILSON†
Stanford University Stanford University Sandia National Laboratories

Abstract

Despite intensive efforts in the area of dynamic graph algorithms, no efficient algorithms are known for the dynamic versions of some basic graph problems such as strong connectivity and transitive closure. We provide some explanation for this lack of success by presenting quadratic lower bounds on the *strong certificate complexity* of such problems, thereby establishing the inapplicability of the only known general technique for designing dynamic graph algorithms, viz., sparsification. These results also provide evidence of the inherent intractability of such dynamic graph problems. Some of our results are based on a general technique for obtaining lower bounds on the strong certificate complexity for a class of graph properties by establishing a relationship with the *witness complexity*.

In many real applications of dynamic graph problems, a certain amount of *lookahead* is available. Specifically, we consider the problems of assembly planning in robotics and the maintenance of relations in databases which, respectively, give rise to dynamic strong connectivity and transitive closure. We exploit the (naturally available) lookahead in these two applications to circumvent the inherent complexity of the dynamic graph problems. We propose a variant of sparsification, viz., *lookahead based sparsification*, and apply it to obtain the first efficient fully dynamic algorithms for strong connectivity and transitive closure.

*Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by 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. This work was performed 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 [4, 5, 6, 12, 14], recently Eppstein, Galil, Italiano, and Nissenzweig [7] presented a general paradigm called *sparsification* that is useful in both designing new dynamic graph algorithms, and speeding up existing ones. The sparsification technique is based on the notion of a *sparse strong certificate* which, informally, 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 graph properties[‡] such as edge and vertex connectivity, bipartiteness, and minimum spanning tree, thereby obtaining efficient dynamic graph algorithms.

In practice, two particularly important dynamic graph problems are *strong connectivity* (in industrial robotics applications [15]) and *transitive closure* (in database applications [18]). Despite intensive efforts, it has proved to be impossible to devise efficient dynamic graph algorithms for these two problems (and a host of others), with or without the use of sparsification. In general, it appears that sparsification is of little help 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 and providing evidence towards the inherent complexity of the dynamic maintenance problem.

Fortunately, many real systems have some form of *lookahead* available, i.e., the dynamic algorithm is provided with information about future updates. We consider in detail a problem in industrial robotics — *assembly planning* — which requires an efficient dynamic algorithm for strong connectivity, and indicate why lookahead is available in this setting. Also, the problem of dynamic transitive closure arises in numerous database applications [18]. This is equivalent to maintaining the transitive closure of a (binary) relation as it undergoes frequent updates, and it is usually desirable to update the transitive closure only after processing batches of updates, leading to the availability of lookahead. While the issue of lookahead is receiving increasing attention in *online* algorithms [2, 10, 11], there has been no work along these lines for dynamic graph algorithms. We initiate the study of lookahead as a means to circumvent the inherent complexity of dynamic graph problems by proposing a *lookahead-based variant of sparsification* and applying it to obtain efficient fully dynamic algorithms for strong connectivity and transitive closure.

In Section 2, we briefly review sparsification in a slightly more general framework than that of Eppstein *et al* [7]. 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. In Section 4, we present a framework for incorporating lookahead into dynamic graph algorithms via a variant of sparsification. We describe the assembly planning problem in Section 5 and develop two algorithms for strong connectivity using our new framework. Finally, in Section 6, we extend this to the transitive closure problem.

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

2 Preliminaries

We will view a graph G as being a set of edges, and the underlying set of vertices will be clear from the context; in general, these graphs will be defined over the same set of n labeled vertices. This interpretation allows us to conveniently apply set operators, such as union and difference, to graphs. We use $|G|$ to denote the number of edges in a graph G.

Eppstein, Galil, Italiano, and Nissenzweig [7] present the sparsification technique in a framework designed for boolean graph properties, although they also provide applications to non-boolean functions such as minimum spanning tree. We review their 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.*

It is easy to verify the following two propositions due to Eppstein, Galil, Italiano, and Nissenzweig [7] hold under the extended definition.

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 $G \cup H$.*

Propositions 1 and 2 together imply that a recursive divide-and-conquer approach can be used for computing the 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. Thus using the transitivity and the monotonicity properties of the sparse strong certificates, 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 has only linear number of edges, 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 devised efficient dynamic algorithms for a host of graph properties including connectivity, minimum spanning trees and bipartiteness. However, sparsification has its limitations: there are many dynamic graph problems which do not yield to sparsification.

Some examples of such graph properties include strong connectivity, maximum matchings, transitive closure and graph search trees. In this section we endeavor to explain this phenomenon by establishing the lack of sparse strong certificates for these properties. We show that each of these properties have instances which require quadratic-sized certificates. We begin with a few definitions.

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 the transitive closure of a graph.

Definition 4 (Critical Graph) *A graph G is called* $\mathcal{P}$-critical *if for every 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 the graph property $\mathcal{P}$ is defined as* $\max_{G \in \mathcal{H}} |G|$.

Let $\mathcal{C}(G)$ denote the set of graphs G' which are strong $\mathcal{P}$-certificates for 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 $\mathcal{P}(G_1 \cup H) \neq \mathcal{P}(G_2 \cup H)$.*

Thus a graph $G' \in \mathcal{C}(G)$ iff 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 $\mathcal{P}$-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 our assumption, G_1 must contain at least one edge not present 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 $|\mathcal{F}| = 2^{g(n)}$, we obtain the desired result.

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

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 X be the set of vertices in H_2 which are reachable from x and let Y be the set of vertices in H_2 that y is reachable from. Clearly, $X \cap Y = \emptyset$, and $x, y \notin X \cup Y$. Let $Z = V \setminus (X \cup Y \cup \{x, y\})$. For any collection of disjoint sets of vertices $V_1, \ldots, V_k$, we define $\mathcal{H}(V_1, V_2, \ldots, V_k)$ as the complete k-partite graph where $(u, v) \in \mathcal{H}(V_1, V_2, \ldots, V_k)$ if and only if u and v do not belong to the same set V_i. Finally, we define H as the graph $\mathcal{H}(\{y\}, Z, Y, X, \{x\}) \setminus H_1$. It is easy to see that $H_1 \cup H$ is strongly connected. On the other hand, we observe that the graph $H_2 \cup H$ is not strongly connected as there is no path from x to y in $H_2 \cup H$. Let $TC(G)$ denote the transitive closure of a digraph G. 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, assume that such a graph H exists. Consider 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 either an edge from a vertex in X to another vertex in X, or an edge from a vertex in Y to another vertex in Y, or an edge from a vertex in 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 of the property of maintaining a maximum matching in a graph is* $\Omega(n^2)$.

This theorem can be extended to the boolean property of having a perfect matching in the graph. The basic idea is to consider again a specific bipartite graph G and show that if the certificate contains an edge not in H, then using the special structure of G, we can always create a distinguishing subgraph for this property. We defer the details to the final version.

We can use techniques similar to the above to establish the following results concerning breadth-first and depth-first search tree maintenance in dynamic graphs; the proofs are omitted.

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 Canonical Hard Instances and a Conjecture

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 6 *Let* $f(n)$ *denote the strong certificate complexity of a property* $\mathcal{P}$, *and* $h(n)$ *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, not in between. More precisely, we make the following conjecture.

Conjecture 1 *Let* $\mathcal{P}$ *be a monotone graph property such that there exists a graph* $G \in \mathcal{P}$ *such that* G *has* $O(n)$ *edges. Then the strong certificate complexity of* $\mathcal{P}$ *is either* $O(n)$ *or* $\Omega(n^2)$.

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, many of these properties 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 of the updates that will be performed in the future. Formally, we say a problem has a lookahead k if at each step, the next k edge operations are fully specified. This lookahead framework encompasses both the off-line setting and a fully-dynamic setting as special cases, as the value of the parameter k is varied from zero to infinity.

In this section, we introduce a new notion of sparsification, namely, *lookahead based sparsification*. The sparsification technique as presented by Eppstein, Galil, Italiano, and Nissenzweig [7] is essentially an edge-sparsification technique where the sparse certificates are used to 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 next k operations 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 graph H (which may possibly be dense in edges).

Definition 8 (Lookahead Certificate) *Given a sequence σ of k operations to be performed on a graph G, a* k-lookahead certificate *of size $(f(k), g(k))$ is a graph H on $f(k)$ vertices with $g(k)$ edges for which there exists a function $\mathcal{F}$, as well as a mapping $\mathcal{E}$ from edge operations in G to edge operations in H, such that $\mathcal{P}(G, \sigma) = \mathcal{F}(G, \mathcal{P}(H, \mathcal{E}(\sigma)))$.*

Intuitively, a lookahead certificate is a graph on a small set of vertices whose edges can encode the structural changes that occur under the sequence of operations known in advance. We now indicate how the notion of lookahead certificates may be used in designing efficient dynamic algorithms. Suppose we are given a sequence σ of operations to be performed on a graph G and we wish to evaluate property $\mathcal{P}$ on graph G after each operation; let $\mathcal{P}(G, \sigma)$ denote the output sequence. Partition the given sequence into $p = \lceil |\sigma|/k \rceil$ sequences of length k each, except possibly the last one. Let σ_i denote the ith sequence and let H_i denote the lookahead certificate for the graph G_i with respect to the sequence σ_i, where G_i is the graph G after the operations in $\sigma_1, \sigma_2, \ldots, \sigma_{i-1}$ have been performed. By the definition of lookahead certificates, we have that

$$\mathcal{P}(G, \sigma) = \mathcal{F}(G_1, \mathcal{P}(H_1, \mathcal{E}(\sigma_1))) \circ \mathcal{F}(G_2, \mathcal{P}(H_2, \mathcal{E}(\sigma_2))) \circ \cdots \circ \mathcal{F}(G_p, \mathcal{P}(H_p, \mathcal{E}(\sigma_p)))$$

where $\circ$ denotes concatenation.

Let G be any n-vertex graph with m edges, and let $T_1(n, m, |\sigma|)$ denote the worst case time complexity of computing $\mathcal{P}(G, \sigma)$ by some dynamic algorithm A, let $T_2(n, m, |\sigma|)$ denote the worst-case time complexity of computing $\mathcal{F}(G, \mathcal{P}(G, \sigma))$ and let $T_3(n, m, |\sigma|)$ denote the worst-case time complexity of computing a $|\sigma|$-lookahead certificate. Then the total cost of the lookahead algorithm is given by

$$\left\lceil \frac{|\sigma|}{k} \right\rceil [T_1(f(k), g(k), k) + T_2(n, m, k) + T_3(n, m, k)],$$

as opposed to $T_1(n, m, |\sigma|)$ which results from a direct application of algorithm A.

5 Assembly Sequencing and Strong Connectivity with Lookahead

Automated assembly planning holds promise 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., [9]).

Assembly sequencing is a subproblem of assembly planning. Given a set of parts and their final positions in a product, 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.

The *blocking graph* is a central structure in assembly sequencing, allowing polynomial-time sequencing in many situations [16, 17]. 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 [15, 16].

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 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 [16]. 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).

5.1 Exploiting Lookahead in Dynamic Strong Connectivity

In principle, in the above application, 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. 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.

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 to verify the strong connectivity from scratch after each edge operation leads to an $\Omega(tn^2)$

solution to this problem.

We may improve the running time by making use of the fact that the sequence of edge operations is partially known in advance at any given time. The idea is to decompose the processing into t/k phases with each phase involving precisely k edge operations, where k is the amount of lookahead available. 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 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 $G_p \cup E^A$.

The following lemmas are crucial for the proof of correctness of our algorithm.

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 $H_{p,i}$.*

We defer the proofs to the final version.

Theorem 7 *The graph $G_{p,i}$ is strongly connected if and only if 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. Also, by Lemma 3, the strong connectivity of $G_{p,i}$ implies that $H_{p,i}$ is strongly connected. To see the other direction, 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 $\mathrm{TC}(.)$ 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 the above algorithm, else it simply uses the naive approach of running the strong connectivity algorithm for each operation in the phase.

Theorem 8 *There is an algorithm for fully-dynamic maintenance of the strong connectivity of a graph at a cost of* $O\left(\min\{m, k^2 + T(n,m)/k\}\right)$ *per operation using a lookahead of size* $k = O(T(n,m)^{1/3})$, *where* $T(n,m)$ *denotes the time complexity of the transitive closure algorithm used.*

In the above algorithm, we continued to recompute the strong connectivity from scratch for each $H_{p,i}$ within each phase p. But observe that the problem of determining strong connectivity of each $H_{p,i}$ within each phase p is identical to our starting problem. This suggests a recursive approach which achieves a significantly improved running time. We defer the proof of the next theorem to the final version.

Theorem 9 *There is an algorithm for fully-dynamic maintenance of the strong connectivity of a graph at a cost of* $O(T(n,m)/n)$ *per operation using a lookahead of size* $\Theta(n)$, *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 [1]. 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 [3] 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 [13] 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.

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 our next application of this technique, we show that it achieves a significant speedup even while using the $O(mn)$ time combinatorial algorithm for transitive closure.

5.2 Practical Implications

In terms of our motivating application, 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 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.

6 Exploiting Lookahead in Transitive Closure

In this section, we briefly sketch the extensions of the techniques from the previous section to transitive closure computation. We borrow the notation from there 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 [8], 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 $O(T(n,m) + kT(k,k^2) + n^2k^2/\log^2 n)$, which is minimized when the lookahead is $k = \sqrt{T(n,m)} \log n / n$. We obtain the following theorem.

Theorem 10 *There is a* combinatorial *algorithm for fully dynamic maintenance of the transitive closure of a graph at a cost of* $O((n\sqrt{mn})/\log n)$ *per operation, using a lookahead of size* $\Theta(\sqrt{m/n} \log n)$.

Contrast this bound with the bound of $O(nm)$ time per operation achieved by the naive algorithm which performs a transitive closure computation at each step. In fact, we can obtain further speed-ups by using fast matrix multiplication instead of the combinatorial transitive closure algorithm.

Theorem 11 *There is an algorithm for fully dynamic maintenance of the transitive closure of a graph at a cost of* $O((n\sqrt{\mathrm{M}(n)})/\log n)$ *per operation using a lookahead of size* $\Theta(\sqrt{\mathrm{M}(n)} \log n/n)$, *where* $\mathrm{M}(n)$ *denotes the time complexity of boolean matrix multiplication.*

These ideas also apply to the single-source transitive closure problem; we defer the details to the final version.

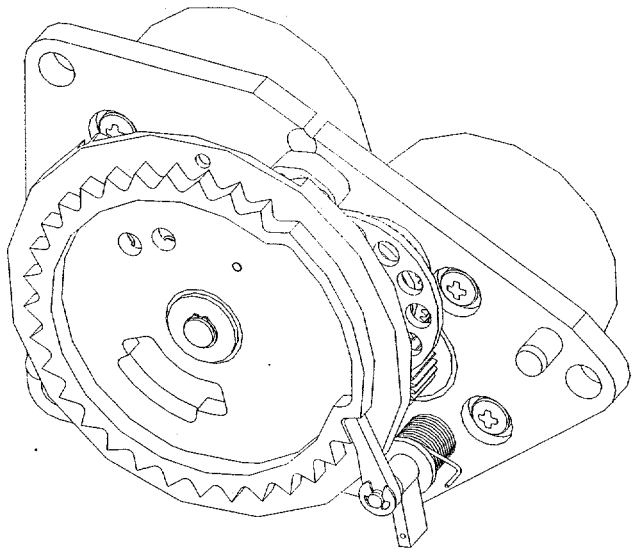

Figure 1: The discriminator assembly

References

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

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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).

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

[10] 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.

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

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

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

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

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

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

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

[18] M. Yannakakis. Graph-theoretic Methods in Database Theory. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (1990), pp. 230–242.

Appendix

A Geometric Assembly Planning

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; however, computing the strong connectedness of a sequence of blocking graphs dominates the algorithm's running time for each class of motion. 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 [16] for further details).

A.1 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.

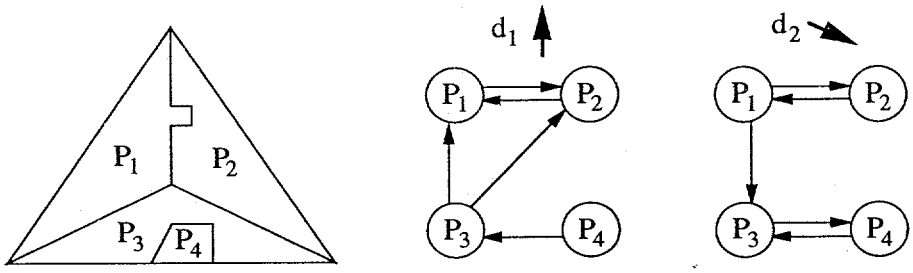

Figure 2: A simple assembly and blocking graphs for two directions of motion

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 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.

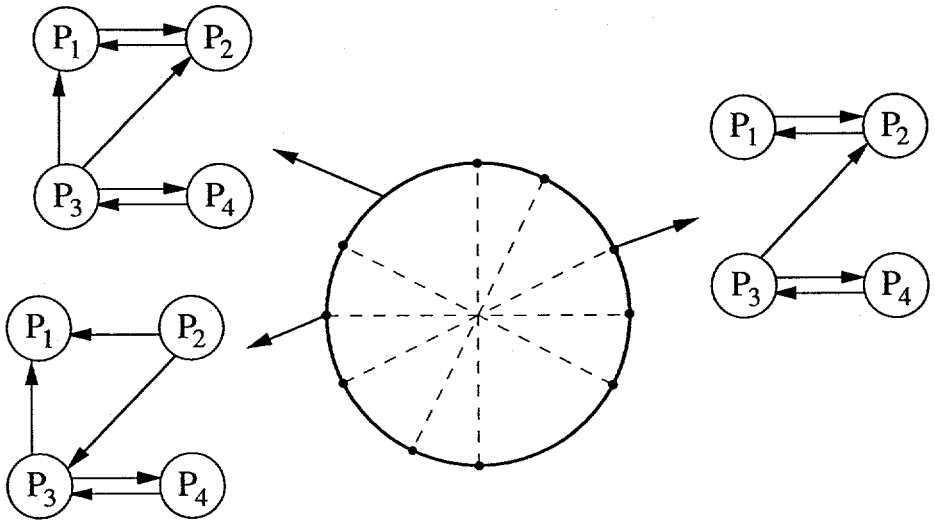

Figure 3: The arrangement on S^1 and some of the blocking graphs for the assembly in figure 2

A.2 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.

A.3 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 [16]. 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. Thus the running time for the local rigid motion disassembly algorithm is $O(c^5 f(n))$.