Efficient Parallel Algorithms for Planar DAGs

Stephen Guattery   Gary L. Miller

May 1995

CMU-CS-95-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Parts of this report appeared in the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '92)

This work was supported in part by the Air Force Materiel Command (AFMC) and the Advanced Research Projects Agency (ARPA) under contract number F19628-93-C-0193.

Keywords: Algorithms, Graph Theory, Graph Algorithms, Parallel Algorithms, Planar Graphs

Abstract

We show that testing reachability in a planar DAG can be performed in parallel in $O(\log n \log^* n)$ time ($O(\log n)$ time using randomization) using $O(n)$ processors. In general we give a paradigm for reducing a planar DAG to a constant size and then expanding it back. This paradigm is developed from a property of planar directed graphs we refer to as the Poincaré index formula. Using this new paradigm we then "overlay" our application in a fashion similar to parallel tree contraction [MR85, MR89]. We also discuss some of the changes needed to extend the reduction procedure to work for general planar digraphs. Using the strongly-connected components algorithm of Kao [Kao93] we can compute multiple-source reachability for general planar digraphs in $O(\log^3 n)$ time using $O(n)$ processors. This improves the results of Kao and Klein [KK90] who showed that this problem could be performed in $O(\log^5 n)$ time using $O(n)$ processors. This work represents initial results of an effort to apply similar techniques to arbitrary planar directed graphs, and to develop efficient algorithms for certain problems encountered in parallel compilation.

1 Introduction

Testing if there exists a path from a vertex x to a vertex y in a directed graph is known as the reachability problem. Many graph algorithms either implicitly or explicitly solve this problem. For sequential algorithm design the two classic paradigms for solving this problem are BFS and DFS. They only require time at most proportional to the size of the graph. Parallel polylogarithmic time algorithms for the problem now use approximately $O(M(n))$ processors, where $M(n)$ is the number of processors needed to multiply two $n \times n$ matrices together in parallel. For sparse graphs the situation is better, though still not optimal with respect to work: Ullman and Yannakakis give a probabilistic parallel algorithm that works in $O(\sqrt{n})$ time using n processors [UY90]. This blow-up in the amount of work for parallel algorithms makes work with general directed graphs on fine grain parallel machines virtually impossible. One possible way around this dilemma is to find useful classes of graphs for which the problem can be solved efficiently. In pioneering papers Kao [Kao93], Kao and Shannon [KS89] [KS93], and Kao and Klein [KK90] showed that the reachability problem and many related problems could be solved in polylogarithmic time using only a linear number of processors for planar digraphs. The planar reachability problem for multiple start vertices is specifically addressed in [KK90]. The methods in that paper involve a series of reductions between related problems; each reduction introduces more logarithmic factors to the running time. In the end it takes $O(\log^5)$ time to solve this problem.

In this paper we give a general paradigm for reducing planar directed acyclic graphs (DAGs) to a constant size. We will show that after $O(\log n)$ rounds of reduction an n-node directed planar DAG will be reduced to a constant size. There have been several reduction rules proposed for undirected planar graphs [Phi89, Gaz91] but this is the first set for a class of directed planar graphs. After we present the rules for reduction it will be a relatively simple matter to "overlay" rules necessary to compute multiple-source reachability.

The results in this report are part of a larger effort to develop a set of reduction rules for arbitrary planar directed graphs (i.e., those with cycles as well as DAGs). We feel that the class of directed planar graphs are important for at least two reasons. First, the class includes several important subclasses including tree and series parallel graphs. Second, the flow graphs for many structured programming languages without function calls are planar. Our goal is to develop the basic algorithmic foundation for a class of planar graphs so that a theory of planar flow graphs could be based on it.

This report presents the details of the reduction technique for the planar DAG case. This case is already quite complicated, and is sufficient to fill a report itself (although we do discuss changes involved in extending the reduction procedure to the general case). Further, we feel that our algorithm for planar DAGs is interesting in its own right. First, it alone is sufficient to improve the computation of many-source reachability by a factor of $\log^2 n$ time by simply using the strong connectivity of Kao [Kao93] (our algorithm for general planar digraphs should remove one further $\log n$ factor). Second, it uses new topological techniques, in particular, the Poincaré index formula. This should be of interest in parallel algorithm design for digraphs.

Throughout the paper we will assume that the graph $G = (V, A)$ is a directed embedded planar graph. If an an embedding is not given we can construct one in $O(\log n)$ time using n processors using the work of Gazit [Gaz91] and Ramachandran and Reif [RR89]. We assume that the embedding is given in some nice combinatorial way such as the cyclic ordering of the arcs radiating out of each vertex.

This paper is divided into seven sections. The second gives the main definitions necessary to define and analyze the directed graph reduction algorithm. The third gives the reduction algorithm for special case of of a planar DAG. The theorems in Sections 4 and 5 show that the reduction algorithm for planar DAGs works in a logarithmic number of reduction steps: Section 4 shows that at any step of the reduction process, a constant fraction of the edges are candidates for removal or contraction; Section 5 shows that a constant fraction of these candidates can be operated on without affecting a set of invariants that we require for the graph structure. The sixth section explains how the reduction procedure can be applied to the many-sources reachability problem and calculates the running time. Finally, in Section 7 we discuss work in progress, including some of the steps necessary to extend this result to the case of general planar digraphs.

2 Preliminaries

2.1 Planar Directed Graphs

We will assume that the reader is familiar with basic definitions and results from graph theory that apply to undirected graphs (see, for example, textbooks such as the one by Bondy and Murty [BM76]).

A **directed graph (digraph)** $G(V, A)$ is a set of vertices V and a set of arcs A. Each arc $a \in A$ is an ordered pair drawn from $V \times V$. We say that arc $a = (u, v)$ is directed from u to v; u is the **tail** and v is the **head** of the arc. We say that an arc is **out of** its tail and **into** its head. An arc a is incident to a vertex v if v is the head or the tail of a. The **degree** of a vertex v is the number of arcs incident to it; we represent this number as $degree(v)$. The **indegree** of a vertex v is the number of arcs that have v as their head; the **outdegree** of v is the number of arcs with v as their tail.

For any directed graph G we can define an undirected graph G' on the same set of vertices in the following way: for each arc (u, v) in G we include an edge (u, v) in G'. We refer to G' as the **underlying graph** of G. In this report we will distinguish between edges and arcs: edges are undirected and lie in the underlying graph, while arcs are directed. When we refer to arcs in G as edges, we are actually referring to the associated edges in G'. (We will use the notation E to represent the set of edges of an undirected graph.)

A **directed path** is a sequence of vertices $(v_0, v_1 \ldots v_k)$ such that the v_i's are distinct (with the exception that we might have $v_0 = v_k$) and for all $0 < i \leq k$ we have the arc (v_{i-1}, v_i) in A. A **directed cycle** is a directed path such that $v_0 = v_k$. A digraph that contains no directed cycles is called a **directed acyclic graph (DAG)**.

For a directed path p that is not a cycle, we define the **rank** of a vertex v on p as the number of vertices on p that precede v.

A **planar directed graph** is a directed graph that can be drawn in the plane in such a way that its arcs intersect only at vertices. There may be a number of different ways to draw a digraph in the plane; any particular way can be specified by giving the cyclic ordering (either clockwise or counterclockwise) of the arcs incident to each vertex. Such a specification is called a **planar embedding** of the digraph.

If the points corresponding to the arcs in an embedded planar digraph are deleted, the plane is divided into a number of connected regions. These regions are called **faces**. The **boundary** of a face is the set of arcs that are adjacent to that face. The **size** of a face is the number of arcs encountered in a traversal of the face's boundary (note that a single arc could be counted more than once in the size of some face). We denote the set of faces by F. (The definitions of these terms are essentially the same for an undirected embedded planar graph.)

In an embedded planar digraph we define **parallel arcs** as two arcs that form a face of size 2. **Parallel edges** in an embedded planar graph are defined in the same way. We can extend this relation by making it transitive; in that case we say that a set of arcs or edges is **mutually parallel**.

There is an important formula developed by Euler (not surprisingly referred to as **Euler's formula** that relates the numbers of edges, vertices, and faces in planar graphs. Euler's formula, which holds for embedded connected planar graphs, is

$$|V| - |E| + |F| = 2. \tag{1}$$

If the graph is also simple (i.e., it has no loops and no parallel edges) and has 3 or more vertices, then each face will have at least three edges in its boundary, and it is easy to prove the following inequality:

$$|E| \leq 3 \cdot |V| - 6. \tag{2}$$

Proofs that these formulas hold can be found in basic graph theory textbooks (e.g., [BM76]). The formula corresponding to (1) (with $|A|$ substituted for $|E|$) holds for embedded planar digraphs that have a connected underlying graph since the orientations of the arcs do not affect the quantities involved. The inequality corresponding to (2) (with $|A|$ substituted for $|E|$) holds for an embedded planar digraph G with a connected underlying graph if G has no loops or parallel edges. Note that this implies that it holds for embedded planar DAGs with connected underlying graphs if the DAGs contain no parallel arcs.

2.2 The Poincaré Index Formula

Let $G(V, A)$ be a connected embedded planar digraph with faces F. We say that a vertex of G is a **source(sink)** if its indegree(outdegree) is zero. The **alternation number** of a vertex is the number of direction changes of the arcs (i.e., "out" to "in" or vice versa) as we cyclically examine the arcs radiating from that vertex. Observe that the alternation number is always even. Thus, a source or a sink has alternation number zero. A vertex is said to be a **flow** vertex if its alternation number is two. It is a **saddle** vertex if the alternation number is 4 or more. Vertex alternations are indicated by asterisks in Figure 1. The alternation number of a face can be defined in a similar way. Here we count the number of time the arcs on the boundary of the face change direction as we traverse its boundary. Thus, a **cycle** face has alternation number zero, a **flow** face has alternation number two, and a **saddle** face has an alternation number greater than two. Face alternations are indicated by asterisks in Figure 2 below. We denote the alternation number of vertex v by $\alpha(v)$, and the alternation number of face f by $\alpha(f)$ (it will be clear from the context whether α refers to a vertex or a face, so we do not distinguish this in our notation).

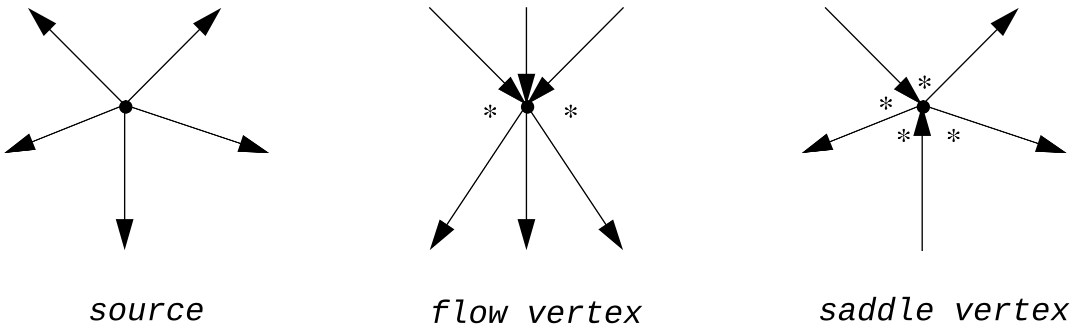


Figure 1: Vertex Types

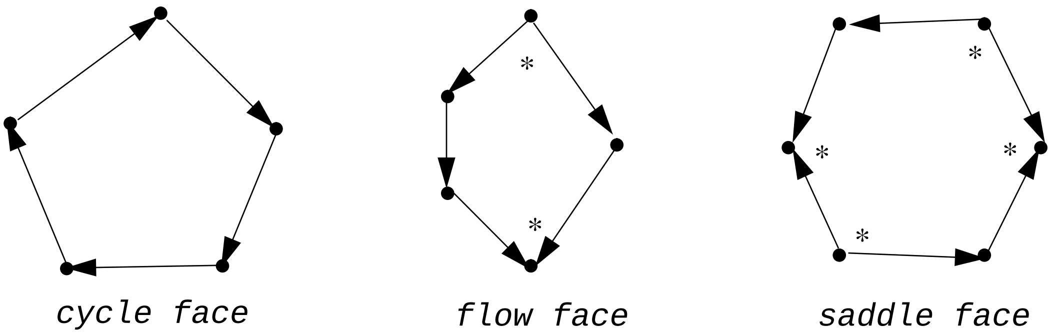


Figure 2: Face Types

A concept related to alternation number is **index**. The index of a vertex v (denoted $index(v)$) is defined as $index(v) = \alpha(v)/2 - 1$. The corresponding definition holds for the index of a face. Once again we do not distinguish between the notation used in these two cases.

Our approach depends on combinatorial arguments based on the following simple but fundamental theorem which we refer to as the **Poincaré index formula**. We show that it follows directly from Euler's formula.

Theorem 2.1 *For every embedded connected planar digraph, the following formula holds:*

$$\sum_{v \in V} index(v) + \sum_{f \in F} index(f) = -2.$$

Proof: Consider the situation at any vertex as we cycle through its incident arcs in order according to the embedding. Each transition from one arc to the next results in exactly one alternation either for the vertex or for the face for which the two arcs lie on the boundary (see Figure 3). If we sum

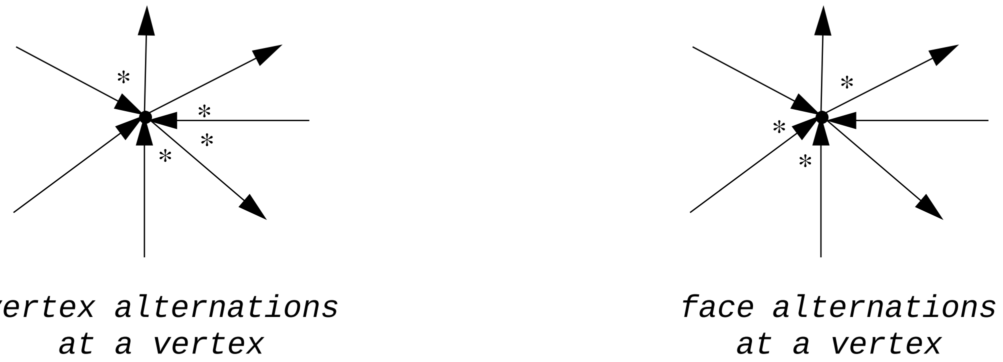


Figure 3: Alternations

the number of alternations over all vertices, we see that the total number of alternations in the graph is equal to the sum of the degrees of all the vertices, which is equal to twice the number of arcs in the graph:

$$\sum_{v \in V} \alpha(v) + \sum_{f \in F} \alpha(f) = \sum_{v \in V} degree(v) = 2 \cdot |A|\,.$$

Dividing by two and applying Euler's formula, we get

$$\sum_{v \in V} \frac{\alpha(v)}{2} + \sum_{f \in F} \frac{\alpha(f)}{2} = |A| = |V| + |F| - 2,$$

which gives us

$$\sum_{v \in V} \frac{\alpha(v)}{2} \;-\; |V| \;+\; \sum_{f \in F} \frac{\alpha(f)}{2} \;-\; |F| \;=\; -2,$$

which with some rearrangement and an application of the definition of index gives us

$$\sum_{v \in V} \left(\frac{\alpha(v)}{2} - 1 \right) + \sum_{f \in F} \left(\frac{\alpha(f)}{2} - 1 \right) = \sum_{v \in V} index(v) + \sum_{f \in F} index(f) = -2.$$

□

This formula is important because it tells us a great deal about the structure of a planar digraph embedding. For example, we have defined various types of faces and vertices in terms of their alternation numbers; this has implications with respect to the contributions of these structures in the Poincaré Index Formula:

- Sinks, sources, and cycle faces each contribute -1. These are the only elements that make negative contributions to the sums in the formula; since the total must be -2, it is clear that every embedded planar digraph must have at least two such elements. For example, a strongly connected planar digraph cannot have any sinks or sources, so it must have two cycle faces.

- Flow faces and flow vertices have index 0 and contribute 0 to their respective sums. There can be an arbitrary number of such elements. It is easy to see that a flow face has two alternations on its boundary, one of which looks like a source with respect to the boundary, the other of which looks like a sink. A useful implication of this fact is that at most one source and at most one sink can lie on the boundary of a flow face.

- Saddle vertices and saddle faces have positive (integer) indices that depend on their alternation numbers and thus contribute positive (integer) amounts to their respective sums. Since the formula total must always be -2, the embedded graph must contain a sink, source, or cycle face for every pair of alternations beyond the first on some saddle.

We will use the formula below to develop invariants and to help us count (for example, we use it to count particular types of arcs).

2.3 Models of Parallel Computation

The reduction algorithm is specified for the **Parallel Random-Access Machine (PRAM)** model of computation. We discuss the algorithm for this model in the cases where memory accesses are allowed to be concurrent read, concurrent write (CRCW). We also assume the ARBITRARY model for concurrent writes (i.e., an arbitrary one of the values being written to a memory location during a concurrent write will end up in that location).

3 Graph Reduction

In this section we introduce a collection of reduction rules and an associated data structure for planar DAGs. The reduction rules allow us to convert a graph into a smaller graph in order to recursively solve our problem. Once the problem is solved for the reduced graph, we can expand the graph out in reverse order and generate a solution for the original graph. In Sections 4 and 5 we show that at each stage the reduction process removes a constant fraction of the arcs; thus, the rules could be implemented as an $O(\log |A|)$-step reduction procedure for planar DAGs. Since we will require our inputs to have no parallel arcs, Inequality (2) in Section 2.1 thus implies that the reduction procedure takes $O(\log n)$ steps (where $n = |V|$ in the original graph). The rules listed below represent an abstraction of the reduction procedure that can be applied with slight variations to implement different algorithms. These variations involve algorithm-specific actions performed as each rule is applied; we will specify such actions in the algorithm descriptions in a later section.

We will assume that our input is a connected, embedded planar DAG G that has no parallel arcs (and hence no parallel edges). We preprocess the graph such that G has the following properties (these properties will remain true throughout the algorithm):

1. G has only flow faces. This can be accomplished by putting a source in each saddle face, and putting an arc from this source to every vertex that is a local source with respect to the saddle face boundary (Figure 4). It is straightforward to show that the number of arcs and hence the number of vertices increases by at most a constant factor.

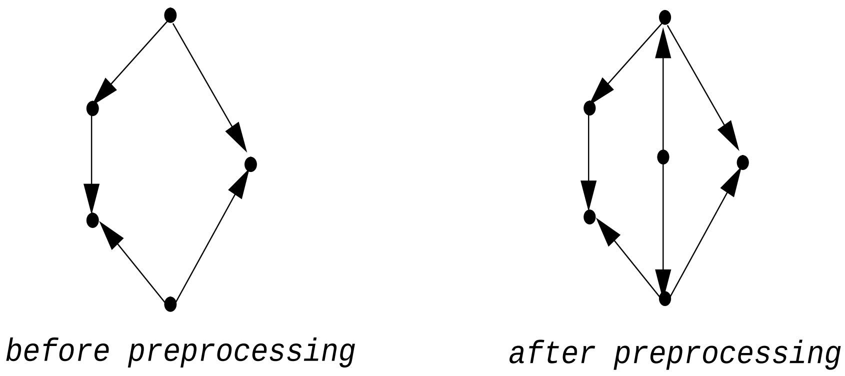


Figure 4: Preprocessing Saddle Faces

2. No vertex has both indegree and outdegree of 1 (i.e., there are no degree-2 flow vertices). Such vertices are considered to be internal vertices of **topological arcs**; such arcs are treated as single arcs with respect to the algorithm, though operations on these arcs may require the internal vertices to perform operations such as splicing connectivity pointers.

It's not hard to see that any connected, embedded planar DAG can be transformed in $O(\log n)$ time so that these conditions are true without changing the reachability of the graph.

3.1 Terminology

In order to simplify the presentation of the reduction rules, we first introduce some concepts and terminology.

Let f be a flow face; then the arcs on its boundary decompose into two paths, a left and a right (we refer to any arc that is both on the left and the right path of a particular face as an **internal** arc). There is also a unique **top** and a unique **bottom** vertex on f. Thus the left path starts at the top vertex and in a counter-clockwise fashion (with respect to the face) goes to the bottom vertex, and the right path goes from top to bottom in a clockwise fashion[1]. A **top(bottom)** arc of f is any arc out of(into) the top(bottom) vertex. An arc may be both a top and a bottom arc for the same face. An arc is referred to simply as top(bottom) if it is the top(bottom) arc for some flow face. We will mark top arcs with "T" and bottom arcs with "B."

In applying the rules we may modify the connectivity of the graph. Therefore we associate flow faces with a data structure that allows us to maintain connectivity information. For each vertex on a flow face that is neither a top or bottom vertex we have a **cross-pointer**, pointing from left to right or right to left. Initially each cross-pointer is set to the bottom vertex. Intuitively, the connectivity on f as determined by its cross-pointers and boundary arcs should be the same as obtained using using arcs and vertices on the boundary of f or those removed from the interior of f by the reduction rules. For each vertex other than top and bottom on a flow face we will also keep the highest and lowest vertex on the opposite side of the face that points to this vertex (initially the high point in will be set to bottom and the low point in will be set to top).

For both the left and right path of each flow face, the top arc will serve as the **leader** of the path (if the top arc is internal it will serve as leader for both sides); each arc will know the two faces common to it and the leaders on those faces. Leader is defined similarly for topological arcs: the leader is the first arc from the original graph in the topological arc (i.e., the arc into the first internal vertex of the topological arc). The rank of the vertices will be maintained on each topological arc.

Using concurrent reads, a leader for each face and topological arc, and the ranking of vertices internal to topological edges, the vertices can now coordinate their actions. For example, cross-pointers can now be tested in constant time to see if they have become forward pointers: simply test if the head and tail are on the same side of the face. (The coordination actions we will use take constant time in the CRCW model.)

We will refer to saddle vertices by their indices. For example, "saddle vertices with index 1" represents the set of saddle vertices with fewest alternations.

Some reduction rules depend on knowing whether an arc is the unique arc into some vertex or the unique arc out of some vertex. We will refer to such arcs as **unique-in unique-out** arcs. Note that it is possible for an arc to be both unique-in and unique-out. In some cases an arc a might not be unique-in, but at the head of a the next arcs in both the clockwise and counterclockwise cyclic ordering may be out-arcs. In that case we say that a is **locally unique-in**; a symmetric definition holds for **locally unique-out**. Note that we will always use "locally" to imply that there is at least one other edge into(out of) the head(tail), though that edge is not adjacent in the cyclic order.

The existence of topological arcs and the introduction of reachability pointers as described above leads to complications in the application of reduction rules. In particular, we need to distinguish certain unique-in and locally unique-in arcs out of a source. We call such an arc a out of a source **clean** if it has the following properties: (1) a has no internal vertices, and (2) for each face

[1]Clockwise and counterclockwise *with respect to a face* can be understood in terms of the dual graph; the clockwise order of arcs on the boundary of a face is the same as the order of the corresponding arcs in the clockwise cyclic order at the dual vertex corresponding to the face.

f that has a on its boundary, there are no pointers across f into the head of a. Clean unique-out and clean locally unique-out arcs into sinks are defined similarly, with the exception that the second condition prohibits pointers across adjacent faces out of the tail of the arc.

We define the operation of **arc contraction** as follows: the contracted arc is removed from the graph, and the head and tail vertices are combined into a single vertex. The cyclic order of the arcs at this new vertex is the cyclic order at the tail with the arcs at the head vertex inserted (in their original order) where the contracted arc was.

3.2 Reduction Rules

We are now ready to list the reduction rules:

[TB Rule] If an arc a is marked both T and B then remove a. If a is topological, it may have crosspointers incident to its internal vertices. A crosspointer into an internal vertex v is adjusted by a process we refer to as **pointer splicing**: the cross-pointer into v is set to point to the vertex pointed to by the cross-pointer out of v on the opposite side of a. The remaining pointers are unchanged. Information on the structure of the face must be updated (e.g., the left and right leaders must be updated), and any new or changed topological arcs must be updated. Pointer updating is shown in Figure 5 (the lighter arrows indicate pointers, the darker ones arcs). **[Degree-1 Rule]**

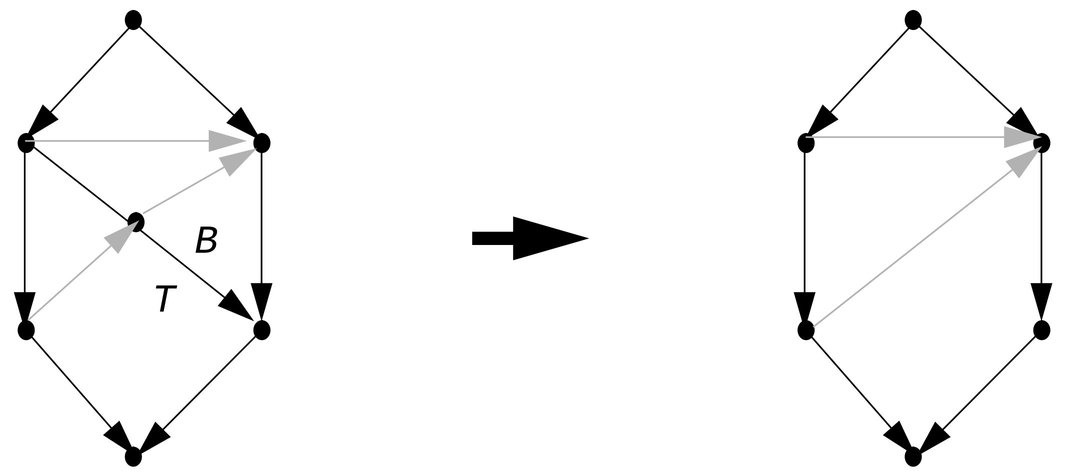


Figure 5: TB Rule Pointer Splicing

If a source or a sink is of degree 1 then remove it and its arc. The leaders on the left and right boundaries of the face are reset if necessary.

[Unique-in(Unique-out) Arc Contraction Rule] If a is a clean unique-in arc out of a source, contract a. The leaders on the affected faces are reset as necessary. The corresponding rule holds for unique-out arcs into sinks.

[Adjacent Degree-2 Sources and Sinks Rule] If a degree-2 source and a degree-2 sink incident to clean arcs are in the configuration shown in Figure 6, remove the source and sink and their arcs as shown. **[Source-Sink-Source (s-t-s)/Sink-Source-Sink (t-s-t) Rule]** Let s be a degree-2 or

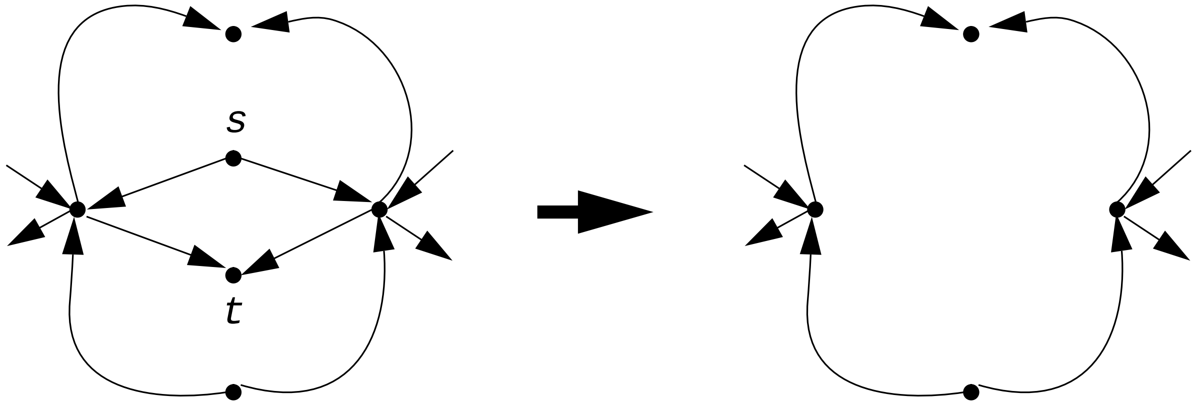


Figure 6: Adjacent Degree-2 Sources and Sinks Rule

degree-3 source incident only to clean locally unique-in arcs. Further, at two of the saddle vertices u_1 and u_2 adjacent to s let there be locally unique-out arcs (respectively a_1 and a_2) such that a_1 (a_2) is adjacent in the cyclic order at u_1 (u_2) to the arc incident to s. If a_1 and a_2 are also incident to distinct sinks t_1 and t_2, take the following actions:

- If s has degree 2, remove the source and its arcs, and combine the two sinks into a single sink (see Figure 7 – since all faces are flow faces, each sink will be at the bottom of one of the two faces on the boundaries of which s lies).
- If s has degree 3, remove the arc out of s common to the two faces on the boundaries of which the two sinks lie, then combine the two sinks into a single sink (Figure 8).

A corresponding rule applies for sinks and adjacent sources. If a large number of vertices are combined into a single vertex, a processor must be selected to represent that vertex. Although this could take time $O(\log n)$, this computation can be done in parallel with the rest of the algorithm without affecting the running time.

[Consecutive Rule] Let s be a source incident to a clean locally unique-in arc a. At the head of a, if the next arcs in both the clockwise and counterclockwise directions are clean locally unique-out arcs into sinks, do the following: remove a, and combine the two sinks into a single sink (see Figure 9). A corresponding rule applies for a sink and adjacent sources.

[Index-1 Saddle Rule] If a source has a clean arc to a saddle vertex of index 1 and if the only other arc into the saddle is a clean arc from another source, then contract one or both[2] of the in-arcs (see

[2] Whether one or both are contracted will be determined by the conflict resolution procedure as discussed in Section 5.1.

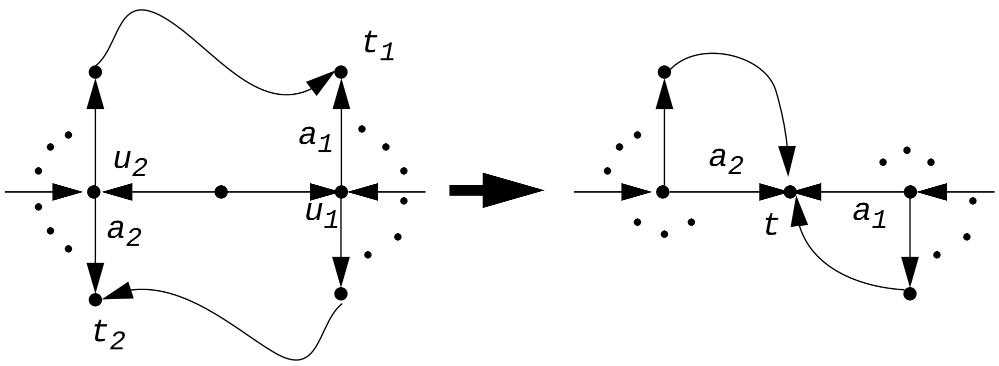


Figure 7: Sink-Source-Sink Rule (Degree-2 Source)

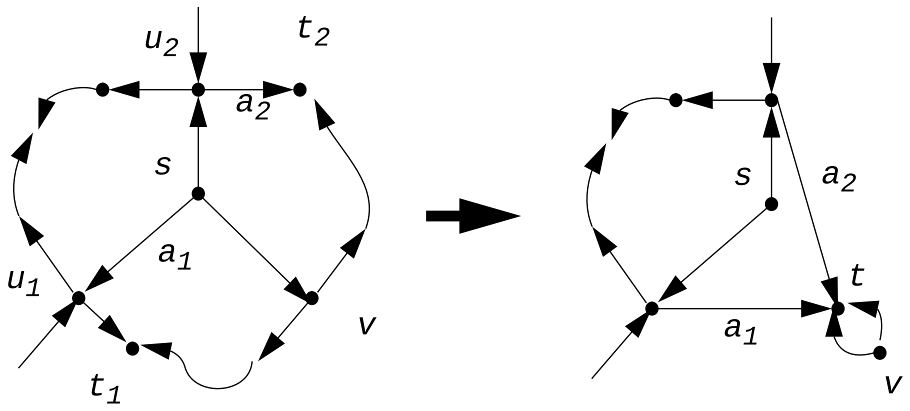


Figure 8: Sink-Source-Sink Rule (Degree-3 Source)

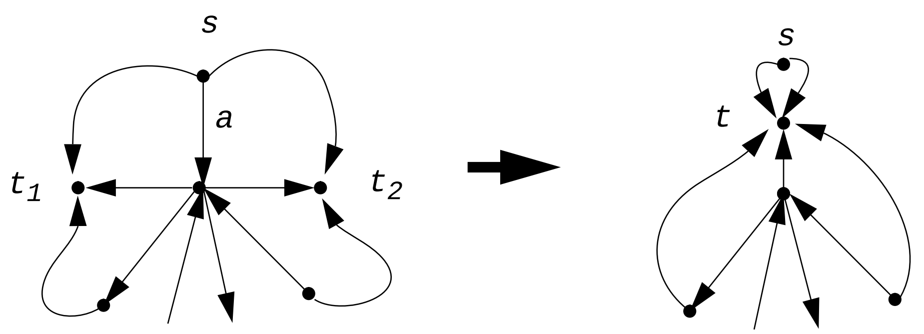


Figure 9: Consecutive Rule

Figure 10). A corresponding rule holds if there are exactly two clean out-arcs to sinks (Fig. 11). As for the s-t-s and t-s-t Rules, if a large number of vertices are combined into a single vertex, a processor must be selected to represent that vertex.

If a source (sink) of degree 2 or 3 has two clean arcs to (from) a single index-1 saddle, note that these two arcs divide the plane into two pieces. Split the graph into two pieces as follows:

- the arcs and vertices in the interior piece of the plane (i.e., the piece not including the exterior face), including the vertex that was an index-1 saddle in the graph prior to rule application; and
- if the source or sink has degree 2, the arcs and vertices in the exterior piece of the plane (including the vertex that was the saddle prior to rule application); if the source or sink has degree 3, the arcs and vertices in the exterior piece of the plane with the third source arc incident to the vertex that was the index-1 saddle. The resulting graph in either case corresponds to the result of deleting the arcs and vertices in the interior piece and then contracting the two arcs incident to the saddle.

Each of the two graphs has strictly fewer arcs than the graph prior to splitting (see Figure 12 below).

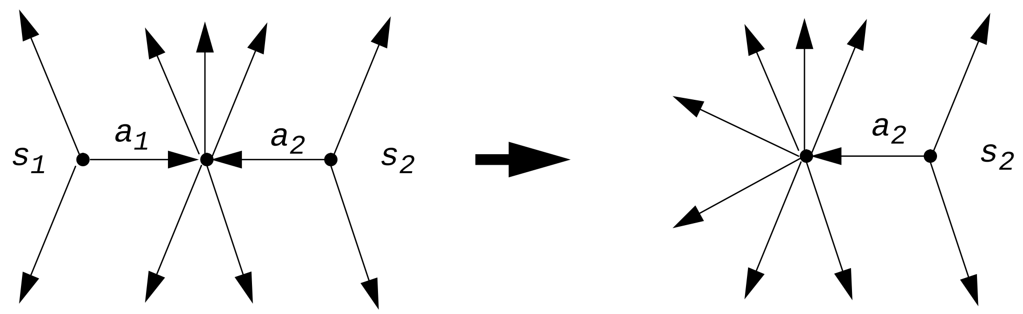


Figure 10: Index 1 Saddle Rule Applied to Sources – only arc a_1 contracted

In the CRCW model it is easy to determine in constant time if the conditions for rule application are met. These conditions can be checked locally in the graph. Ignoring the time to do conflict resolution for now, rule applications can be done in constant time.

3.3 Cleaning Up the Graph

Many of the rules above require that the arcs involved be clean. Arcs in the configurations corresponding to particular rules will not necessarily be clean, however. Therefore we introduce a parallel algorithm for cleaning up arcs out of sources(into sinks) that runs in constant time in the CRCW model. The cleanup algorithm will be run as a subroutine in the reduction algorithm. Because the cleanup algorithm can change the structure of the graph, we may require it to preserve

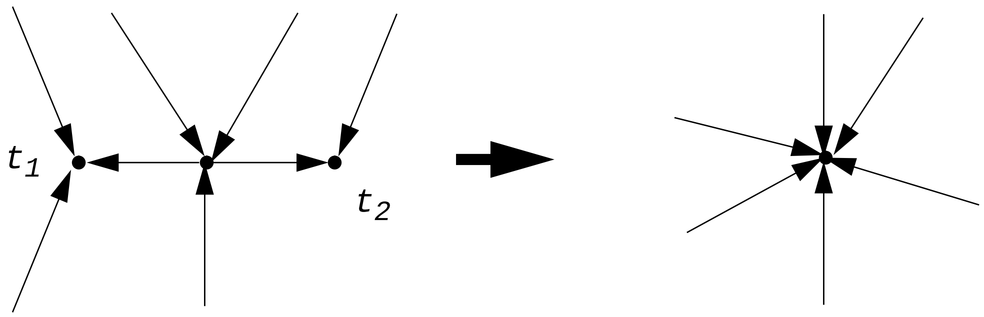


Figure 11: Index 1 Saddle Rule Applied to Sinks – both arcs contracted

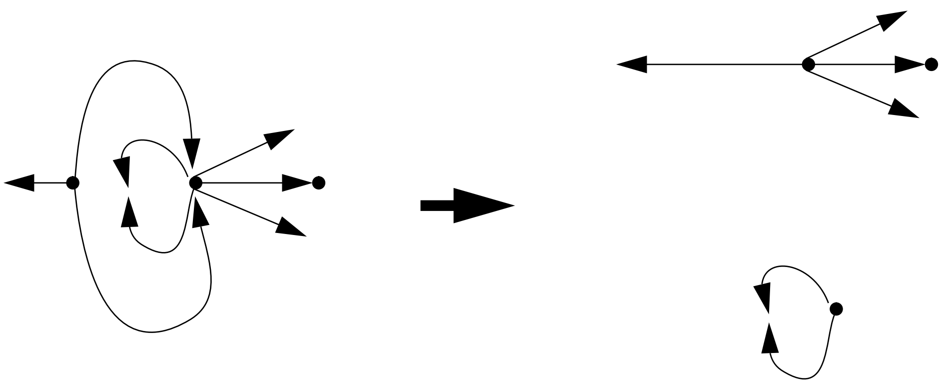


Figure 12: Index 1 Saddle Rule – Single Degree-3 Source Case

some invariant specific to the problem we are solving (e.g., in the case of many-source reachability the invariant will be that the vertices in the current version of the graph that are reachable from one of the original sources are either marked as reachable or have a path consisting of pointers and arcs from some vertex with an active mark). Applying cleanup with respect to the invariant will add computation to the cleanup algorithm; in general, we try to choose an invariant in such a way that it doesn't increase the asymptotic running time of the basic cleanup algorithm.

We do not clean up all sources and sinks. To insure that cleanup doesn't take too long (i.e., cleanup activities other than application-specific processing should take constant time), we will clean up only sources (respectively sinks) of degree less than or equal to a constant d (to be specified later) that are incident only to unique-in or locally unique-in (respectively unique-out or locally unique-out) arcs. Note that such sources and sinks are not incident to parallel arcs. We will explain the rationale for these restrictions in Section 4; intuitively, the conditions given here will allow us to argue that we can remove a number of arcs proportional to the number of sources and sinks: the reduction rules allow operation on parallel arcs (such arcs will be marked both T and B) and clean arcs at sources and sinks. Thus we expect to find an arc to operate on for each clean source or sink; we'll show later that the number of sources and sinks with no incident operable arc is small.

3.4 The Cleanup Algorithm

We define the **frontier** of a source s as the set of vertices at the heads of the arcs out of s. The frontier of a sink t is the set of vertices at the tails of the arcs into t.

The cleanup algorithm consists of several steps. We note that application-specific processing can be added prior to or after any step, and defer further discussion to Section 6, where we discuss a particular application in detail.

At the first cleanup step, each source (respectively sink) determines if it has degree less than or equal to d, and if so, whether all incident arcs are either unique-in or locally unique-in (respectively unique-out or locally unique-out). Any source or sink not meeting these conditions drops out of the cleanup algorithm.

At the second cleanup step, for each arc out of each source still involved in the cleanup we want to find the highest vertex on that arc (including its head) that can be reached from some other frontier vertex. Here "highest" means closest to the source, and "reached" means there exists a path of pointers from the frontier vertex to the high point, where each pointer has as its head and tail some vertex that lies between the source and the frontier (inclusive of frontier vertices). We also need to know which frontier vertex can reach this high point, and whether the pointer path proceeds in a clockwise or counterclockwise direction relative to the source. These things can be determined by following the high pointer chains out of each frontier vertex first in the clockwise direction and then in the counterclockwise direction. Note that if more than one frontier vertex can reach the high point with respect to a single direction (e.g., clockwise around the source), we will choose the one at the greatest distance in terms of the cyclic ordering. The high point can be determined by comparing the location of the clockwise and counterclockwise high points relative to the rank ordering along the topological arc. If the same high point can be reached from both the clockwise and counterclockwise directions, the tie can be broken arbitrarily.

The case for sinks is symmetric. For each arc into each sink still involved in the cleanup we want to find the lowest vertex on that arc (including its tail) that can reach some frontier vertex. Here "reach" will mean there exists a path of pointers from the low point to the frontier vertex, where each pointer has as its head and tail some vertex that lies between the frontier and the sink (inclusive of frontier vertices). Again, we also need to know which frontier vertex can be reached from this low point, and whether the pointer path proceeds in a clockwise or counterclockwise direction relative to the sink. These things can be determined by following the chain of low pointers in reverse order, first in the clockwise direction and then in the counterclockwise direction. Note that if more than one frontier vertex can be reached from the low point with respect to a single direction (e.g., counterclockwise with respect to the sink), we will choose the one at the greatest distance in terms of the cyclic ordering. The low point can be determined by comparing the location of the clockwise and counterclockwise low points relative to the rank ordering along the topological arc. If the same low point can reach the frontier in both the clockwise and counterclockwise directions, the tie can be broken arbitrarily.

Note that for any cleaned source we must have at least one arc a out that has no high point. Similarly, for any cleaned sink there must be at least one arc b such that b has no low point. To see this for sources, note that if an arc a has a high point there is another frontier vertex v that has a path to the head of a consisting of the pointer path to the high point plus the segment of a between the high point and the head. We can construct a directed graph consisting of the frontier vertices for this source and arcs representing the existence of a path from one frontier vertex to another. Since the original graph is a DAG, the graph we have constructed must be acyclic. But if every arc out of the source has a high point, then every frontier vertex in this graph must have an arc in, which contradicts the fact that it is acyclic.

At the third step we realign the graph as shown in Figure 13 below (the presentation here is in terms of sources; the actions for sinks are symmetric). First consider arcs that are reachable from

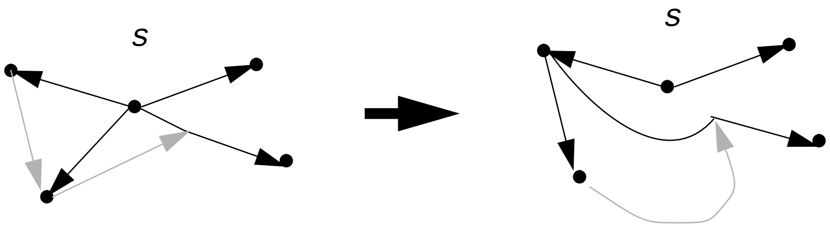


Figure 13: Cleanup: Realignment at a Source

another frontier vertex: Each arc (if the high point is a frontier vertex) or arc segment (if the high point is internal to a topological arc) from the source to the high point is replaced by an arc (or, if internal, an arc segment) from the most distant frontier vertex. If a frontier vertex reaches multiple high points, the cyclic order of the new arcs at the frontier vertex is same as the cyclic order of the deleted arcs to those vertices at the source. All pointers to vertices at or below the high point are retained.

For arcs that are not reachable from another frontier vertex, if the arc is topological, replace the arc with an arc containing no internal vertices (i.e., all internal vertices between the source and the high point are deleted).

Each source and sink that has been cleaned up is now marked as "cleaned up". Each arc out of a cleaned source or into a cleaned sink is marked as "cleaned".

It is easy to verify the following claims: Every cleaned source has at least one arc out; every arc out of a cleaned source is clean. Likewise every cleaned sink has at least one arc in; every arc into a cleaned sink is clean. All the resulting faces are flow faces. The number of arcs and vertices in the graph does not increase. The reachability for every vertex remaining in the graph is unchanged in the following sense: Let u and v be any two vertices that are not internal to topological arcs. If there is a path of arcs and crosspointers from u to v prior to cleanup, then there is such a path after cleanup.

In order to avoid any conflicts between arcs common to both a source and a sink these steps can be run twice, once for sources and once for sinks.

These steps could be implemented in a number of ways. Because the degree of any cleaned source or sink is bounded by a constant and because we can use the leader of each topological arc to keep track of bookkeeping information, the cleanup algorithm can be programmed to execute in a constant number of steps in the CRCW model provided that any application-specific processing added will execute in constant time.

3.5 Overview of the General Reduction Process

The general algorithm for reducing an embedded planar DAG can now be stated:

1. Preprocess the graph to make it consistent with our invariants.
2. Main Reduction Loop: While there are arcs left in the graph, repeat the following sequence of steps:
   - Clean up the current graph, performing any application-specific actions where needed.
   - Apply the reduction rules in the order they're listed in Section 3.2. Application-specific processing may be required as each rule is applied.
3. Perform any application-specific processing needed prior to the expansion phase.
4. Reconstruct the graph by reversing the steps in the reduction process (note that this requires that we have stored, in order, all changes made during the reduction process).

This process will take constant time given that the conflict resolution steps noted take constant time using randomization in the CRCW model; the deterministic algorithm takes time $O(\log^* n)$ for each reduction step. The proof that the graph will be reduced by applying this process $O(\log n)$ times (thus giving an $O(\log n)$ time randomized algorithm or an $O(\log n \ \log^* n)$ deterministic algorithm) is presented in the rest of the report.

4 Operability Lemmas

In the next two sections we prove that the reduction procedure given above works in $O(\log n \log^* n)$ time using $O(n)$ processors, provided that the application-specific processing takes at most constant time per reduction phase. (We will use the convention that $n = |V|$ throughout the rest of this section.) The main result of this section is that at each pass through the main reduction loop a constant fraction of the arcs are candidates for removal (we refer to such arcs as **operable**). We start with two preliminary lemmas.

Lemma 4.1 [Flow Face Operability] *An arc a between two flow faces is operable if it is neither unique-in, locally unique-in, unique-out, nor locally unique-out.*

Proof: Such an arc a must have an adjacent out-arc in the cyclic ordering at its tail vertex, which must be the top vertex of one of the flow faces. Therefore a is a T arc. A symmetrical argument shows that a is also a B arc. Thus, a is operable by the TB rule. $\Box$

Lemma 4.2 [Unique-In and Unique-Out Arc Count] *In a connected, embedded planar DAG consistent with our invariants the number of unique-in and unique-out arcs not incident to degree-1 vertices is less than or equal to $2/3$ the number of arcs in the graph.*

Proof: We note that in an embedded DAG the unique-in arcs (respectively unique-out arcs) form a forest. For the purposes of this proof, we use the term **unique-in(unique-out) tree** to refer to a maximal subgraph of such a DAG that consists of a tree induced by unique-in(unique-out) arcs in the DAG. The unique-in(unique-out) tree may contain vertices that had degree 1 in the original DAG; if these vertices and their incident arcs are deleted from the unique-in(unique-out) tree, the **trimmed unique-in(unique-out) tree** results. We will start by counting the number of arcs to which each trimmed unique-in tree is incident in the current graph G that either 1) are neither unique-in nor unique-out, or 2) are into degree-1 vertices, and proving that this number is greater than the number of arcs in the tree (the proof for unique-out trees is symmetric).

We first claim that every leaf v of a trimmed unique-in tree must have at least two arcs out in G, each of which either is a unique-in arc to a degree-1 vertex or is neither unique-in nor unique-out. To see that there must be two or more arcs out, note that if v were of degree 1 in G, that would contradict the fact that the tree is trimmed; if v were of degree 2, the second arc would have to be an arc out, and v would be a degree-2 flow vertex, contradicting the conditions of the lemma. We further claim that these arcs out of v must either be unique-in arcs to degree-1 vertices or be neither unique-in nor unique-out. Since there are two out-arcs, they can't be unique-out; if they are unique-in they must be to degree-1 vertices or we contradict the assumption that v is a leaf of a trimmed unique-in tree, which by definition is maximal. Therefore the claim must hold.

Next we will pair each arc in the trimmed tree with a distinct arc a in G out of some tree vertex v such that a either is into a vertex of degree 1 or is neither unique-in nor unique-out. Note that each tree arc must be either into an internal node of the tree or into a leaf node. We pair each arc into a leaf v with one of the arcs out of v in G; this leaves one additional arc out of each leaf. To handle internal nodes, we introduce the following terminology: if an internal node has exactly one tree arc out, we call it a *path node*; otherwise it is a *branch node*. (For our purposes we will not count the root as an internal node, though it makes only minor technical differences in the statement of the following.) Each path node in a unique-in tree must have at least one arc out in G that either is incident to a degree-1 vertex or is neither unique-in and nor unique-out; we pair one such arc with the unique tree arc into the path node. The only arcs we still have to pair up are those into branch nodes, which we can associate with distinct arcs from the set of arcs out of the leaves as follows: The number of leaves in a tree is easily shown to be greater than the number of branch nodes. Therefore, since we have exactly one unique-in arc into any branch node, we have fewer arcs into branch nodes than we have arcs left at the leaves. Thus all tree arcs have been paired as claimed. The arcs we've associated with each trimmed unique-in tree arc are clearly distinct, and, since we have only counted arcs out of trimmed unique-in trees, no arc we've counted could be counted for more than one such tree. Therefore there is at least one distinct arc of one of our two types for every unique-in arc in the graph that is not incident to a vertex of degree 1. This completes the argument.

By a symmetric argument, there is either a distinct unique-out arc out of a node of degree 1 or a distinct neither-unique-in-nor-unique-out arc for every unique-out arc in the graph that is not incident to a vertex of degree 1. To finish the proof, we observe that each neither-unique-in-nor-unique-out arc out of a unique-in tree could also be an arc into a unique-out tree; thus, in the worst case we might count each of these arcs twice. In that case the number of arcs we've found is at least $1/3$ the number of arcs in the graph, from which the lemma follows. □

We can now state the main lemma of this section:

Lemma 4.3 [Operability Lemma] *In any connected, embedded planar DAG that has been cleaned up and that is consistent with our invariants, a constant fraction of the arcs are operable.*

Proof: The lemma follows immediately from Lemmas 4.4 and 4.8 below, which prove the result for the cases in which the number of sources and sinks is less than $n/14$ and greater than or equal to $n/14$ respectively. □

Before proving these two lemmas, we'll give brief sketches of the proofs. Lemma 4.4 deals with the case in which the number of sources and sinks is less than a specified fraction of the number of vertices in the graph. Its proof proceeds by showing that there are many arcs that either are unique-in or unique-out and incident to degree-1 sources and sinks, or are neither unique-in nor unique-out. This follows from the Unique-In, Unique-Out Arc Count Lemma (Lemma 4.2 above). This isn't quite enough to prove Lemma 4.4, however; we must then show that most of the non-unique-in, non-unique-out arcs are neither locally unique-in nor locally unique-out. This follows from the Poincaré Index Formula and the conditions of the lemma. By the Flow Face Operability Lemma (Lemma 4.1 above), this is sufficient to show that a constant fraction of the arcs in the graph are operable by the TB Rule.

Lemma 4.8 covers the case in which the number of sources and sinks is at least a specified fraction of the vertices in the graph. We prove it using a counting argument. First we show that a high degree source or sink v (i.e., a source or sink with degree greater than the constant d introduced in Section 3.3) either is incident to a TB arc or is uncommon in the sense that the total number of such sources and sinks is less than a constant fraction of the total number of sources and sinks in the graph. Next we show that at least a constant fraction of the sources and sinks with degree $\leq d$ are incident to an operable arc. This is clearly true for such sources and sinks that are either degree-1 or incident to a TB or arc. Any other such sources or sinks will be processed in the cleanup phase. We will show that at least a constant fraction of the cleaned sources and sinks are incident to an operable arc by a counting argument based on the Poincaré Index Formula. We can then show that, excluding parallel arcs, a constant fraction of the arcs are operable, and since all parallel arcs are both T and B (and hence operable), the lemma follows.

We now proceed with complete proofs:

Lemma 4.4 *In any connected, embedded planar DAG consistent with our invariants and in which the number of sources and sinks is less than $n/14$, $1/6$ of the arcs are operable.*

Proof: To prove this lemma we show that in graphs meeting the stated conditions there are many arcs that either are incident to degree-1 sources or sinks, or are operable by the TB Rule.

Let k be the number of sources and sinks in the graph. To make the proof easier to read, we will use the following notation to refer to specific sets of arcs:

- A_1 will be the set of arcs that are incident to degree-1 vertices.
- A_2 will be the set of arcs not in A_1 ($A_2 = A \setminus A_1$).
- A_3 will be the set of arcs in A_2 that are neither unique-in nor unique-out. We can restate Lemma 4.2 as follows:
$$|A_3| + |A_1| \geq \frac{|A|}{3}.$$
- A_4 will be the set of operable arcs in A_3.
- A_5 will be the set of inoperable arcs in A_3 ($A_5 = A_3 \setminus A_4$).
- A_{op} will be the set of all operable arcs.

Before proceeding, it's useful to recall that graphs meeting our invariants have no degree-2 flow vertices (such vertices become parts of topological arcs), so every vertex other than a source or sink has degree 3 or more. Thus the number of arcs in the graph is at least $3(n-k)/2 + k/2 = 3n/2 - k$. Since all arcs in A_1 are operable, we will give a lower bound on $|A_1| + |A_4|$, which is also a lower bound on the number of operable arcs.

To get a lower bound on the size of A_4, we first note that the graph only has flow faces, so every arc in A_3 lies between two flow faces. Thus by Lemma 4.1 an arc in A_3 will be operable if it is not locally unique-in or unique-out. Recall that a locally unique-in or unique-out arc must be incident to a saddle vertex.

We now apply the Poincaré Index Formula. We have no cycle or saddle faces, so we only need to consider the indices of sources, sinks, and saddle vertices. Since sources and sinks each contribute -1 to the sum and saddles each contribute a positive amount, the total number of saddles is less than or equal to $k - 2$. The formula implies that for each source or sink beyond the first two there are two alternations on some saddle vertex; there are also two additional alternations per saddle vertex. Thus, the graph can have at most $4k - 8$ alternations at saddles vertices. We will associate each alternation with a single arc in the following way: A vertex alternation is associated with a pair of arcs; associate the alternation with the second arc of the alternation with respect to the cyclic ordering of arcs around the saddle vertex (we refer to this arc as the one "following the alternation"; the first arc of the alternation "precedes the alternation"). Note that each locally unique-in or unique-out arc must have an alternation associated with it: such arcs are by definition immediately preceded and followed by alternations. Since each inoperable arc in A_3 is locally unique-in or unique-out, each has an alternation associated with it. Each alternation is associated with exactly one arc, so there will be at most $4k - 8$ inoperable arcs in A_3. This gives us an upper bound on the size of A_5; since A_5 and A_4 partition A_3, we have

$$|A_4| = |A_3| - |A_5| \geq |A_3| - 4k + 8 > |A_3| - 4k.$$

We can now use the fact above with Lemma 4.2:

$$A_{op} \geq |A_1| + |A_4| > |A_1| + |A_3| - 4k \geq \frac{|A|}{3} - 4k.$$

The lemma follows when we substitute using the condition that $k < n/14$ and the fact that $|A| \geq 3n/2 - k$:

$$A_{op} > \frac{|A|}{3} - 4k > \frac{n}{2} - \frac{13k}{3} > \frac{n}{6}.$$

□

In order to prove Lemma 4.8 and thus complete the proof of Lemma 4.3, we first need to introduce some terminology and preliminary lemmas. We will assume that the graph has been cleaned up.

For analysis purposes we associate a value of i with each saddle vertex, where i is equal to the index of that saddle vertex. This value is distributed equally among the alternations at the saddle; each alternation gets $i/2(i+1)$. The alternations assign their values to sources or sinks in the following way:

- Value is assigned only to cleaned sources with only locally unique-in arcs out, or to cleaned sinks with only locally unique-out arcs in. We refer to such sources and sinks as **eligible**.
- Value from a particular saddle vertex is assigned only to eligible sources (or eligible sinks) that are the tails (respectively heads) of arcs incident to that saddle (i.e., only to eligible sources and sinks at distance 1 from that saddle).
- If only one source or sink can be assigned value from a particular saddle, that source or sink receives that saddle's full value. If value from a saddle can be assigned to more than one source or sink, it is done so in the following way: for each eligible source or sink at distance 1 from this saddle, count the number of alternations between it and the next such eligible source or sink in both the clockwise and counterclockwise directions around the saddle. The source or sink is assigned the value for half that number of alternations.

Clearly each saddle vertex assigns a total value of either 0 or its index to sources and sinks. Note that the minimum value that an eligible source or sink can be assigned per locally unique-in or locally unique-out arc is $1/4$.

We refer to the total value assigned to a source or sink as the **value** of that source or sink. Under certain conditions presented in Lemma 4.6 we will allow particular sources or sinks to transfer their value to other sinks or sources. This transfer will be done in such a way that the total value summed over all sources and sinks will remain constant.

We will call a source or a sink with a value of $9/8$ or greater **uncommon**; other sources and sinks are **common**. In the arguments below, we'll associate a distinct operable arc with each common source and with each common sink. Since each such arc can be associated with at most one common source and one common sink, this will prove that the number of operable arcs is proportional to the number of common sources and sinks.

Lemma 4.5 *In an embedded planar DAG with a total of k sources and sinks, more than $k/9$ of the sources and sinks are common.*

Proof: It follows from the remarks above and the Poincaré Index Formula that the total value that can be assigned by all alternations at all saddle vertices is bounded above by $k-2$. Each uncommon source or sink gets value greater than or equal to $9/8$. If $8/9$ of the sources and sinks were uncommon, their total value would be greater than or equal to $(8k/9) \cdot (9/8) = k$, which is greater than the total value available for assignment. □

Lemma 4.6 *In an embedded planar DAG meeting our invariants, every source incident only to clean locally unique-in arcs is either uncommon or at the tail of an operable arc. Similarly, every sink incident only to clean locally unique-out arcs is either uncommon or at the head of an operable arc.*

Proof: The sources and sinks meeting the conditions of the lemma are the eligible sources and sinks as described above. We will argue on the basis of the degree of the eligible source or sink. Note first that if an eligible source or sink is of degree 1, the incident arc is operable by the Degree-1 Rule. Further, if an eligible source or sink is of degree 5 or greater, it is uncommon (the minimum value that an eligible source or sink can get from each adjacent saddle vertex is $1/4$). If an eligible source or sink has degree 3 or 4, then either one of the incident arcs meets the conditions for removal by the Consecutive Rule or it is uncommon (an eligible source or sink will get the value of at least 3/2 alternation from any saddle vertex where the Consecutive Rule doesn't apply). Thus we only need to prove that the result holds for eligible sources and sinks of degree 2 to complete the proof. We will prove the result for the case of sources; the proof for sinks is symmetric. To simplify the arguments below, we refer to an eligible sink t as **adjacent** to an eligible source s at a saddle vertex u if t is incident to arc a_t, s is incident to arc a_s, and a_t and a_s are adjacent in the cyclic order at u.

For eligible sources of degree 2 where each arc out is incident to a different saddle, we have the following cases:

- There are no adjacent eligible sinks at either saddle vertex. In this case the source gets the value of at least 4 alternations. If either saddle has index greater than 1 then the source is uncommon (recall that the value of an alternation at a saddle of index i is $i/2(i+1)$). If both saddles are index 1 and the source has value 1, then each saddle must have an arc in from another eligible source. In this last case the Index-1 Saddle Rule will hold and the source has an operable arc out.

- There is a single adjacent eligible sink (i.e., an eligible sink is at the end of exactly one edge out of one saddle). In this case the source will get the value of at least three and one-half alternations. There are three subcases: First, if both saddles have index of 2 or greater the source will be uncommon. Second, if the saddle with no adjacent eligible sinks has index 1, then either there is another eligible source with an edge into that saddle (in which case the Index-1 Saddle Rule will hold and the source we are considering will have an operable arc out) or the source gets the value of all alternations at that saddle (in which case the source is uncommon). Third, the saddle without the adjacent eligible sink has index greater than 1 and the saddle with the adjacent eligible sink has index 1. Again, at the index-1 saddle either there is a second eligible source with edge into the saddle (in which case the Index-1 Saddle Rule applies at the source under consideration) or the source gets a value of $1/2$ from this saddle and is uncommon.

- There are at least two adjacent eligible sinks. In this case there are two possibilities. If two of the sinks are distinct then either the t-s-t Rule or the Consecutive Rule will hold and the source will have an operable arc out. Otherwise we have the situation shown in Figure 14 below. There are numerous subcases to consider. In the first two the source has an operable

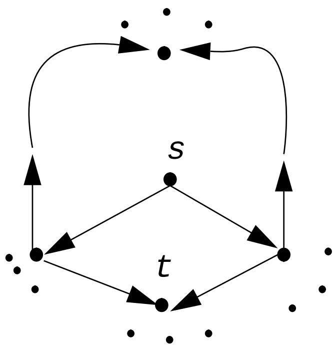


Figure 14: Degree-2 source with common adjacent eligible sink

arc out:

  - The sink has degree 2. Then the Adjacent Degree-2 Sources and Sinks Rule applies and the source has an operable arc out.
  - The sink has degree 3 or greater and one saddle has index 1 and another eligible source with an arc in. Then the Index-1 Rule applies and the source has an operable arc out.

In the remaining subcases there are no rules that apply at the source and we must show it is uncommon (we will refer to this as the problem source configuration:

  - the source has degree 2 and the arcs out are incident to different saddles;
  - a single eligible sink of degree 3 or greater is adjacent at each saddle;
  - and neither adjacent saddle has index 1 and is also adjacent to another eligible source.

). Note that such a source has a value of at least 1 (it will get two alternations from saddles of index 1 and at least $3/2$ alternations at saddles of index 2 or greater). In some of these subcases we transfer value between sources and sinks as mentioned above. In all cases where we transfer value from sinks to sources, the sources must be in the problem source configuration. When a sink transfers value, it divides the value equally among all adjacent sources in such a configuration.

- The sink is operable. In this case we can transfer all its value. The sink has a value of at least degree$(t)/4$; since the number of problem configurations any sink can be involved in is less than or equal to its degree, each source in a problem configuration with this sink will get additional value of at least $1/4$, making it uncommon.
- The sink t has degree 3 or greater and is not operable. In this case we transfer value equal to $1/4$ from the sink to the source, making the source uncommon. We will show below that such a sink has sufficient value to transfer $1/4$ to all such sources to which it is adjacent and still remain uncommon.

For eligible sources of degree 2 where both arcs are incident to the same saddle, we have the following cases:

- If the saddle has index 1, then the Index-1 Rule holds and the arcs out are operable.
- If the saddle has index 2 and there are no adjacent eligible sinks with respect to the source, then the source gets at least 4 alternations and is uncommon.
- If the saddle has index 2 and there is at least one eligible adjacent sink, we have two subcases. The first is as follows: We refer to the two arcs out of the source as a_1 and a_2 respectively. Note that the cyclic order around the saddle is divided into two segments, one between a_1 and a_2 and the other between a_2 and a_1 in the clockwise cyclic order at the saddle. Since the source is eligible, two of the alternations must fall in one segment and four in the other. We note that there can be an adjacent eligible sink in the segment with two alternations only if it is a degree-1 sink that is adjacent to both a_1 and a_2. In this case the Consecutive Rule will apply. This is shown in Figure 15 below.

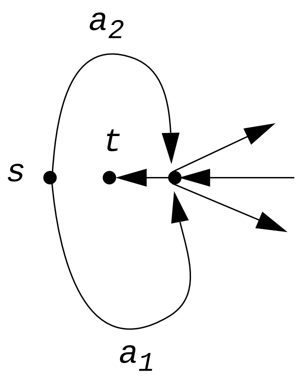


Figure 15: Adjacent eligible sink t in two-alternation segment

- The only remaining case for an index 2 saddle is the case that there are adjacent eligible sinks on the segment with 4 alternations. Note that if there is not an eligible sink adjacent to both a_1 and a_2, then the source will get at least $7/2$ alternations and will be uncommon.

Otherwise, there must be an arc to an eligible sink adjacent to each arc out of the source. Further, since the sink at the end of each such arc lies on the same face as the source, it must be a single sink (there is at most one sink on any flow face). We can treat this situation by looking at the degree of the sink:

  – If the sink is of degree 2, then the Adjacent Degree-2 Source and Sink Rule applies, and the arcs out of the source are operable.
  – If the sink has degree 3 or greater, then we are in a variant of the problem source configuration, and we transfer value of $1/4$ from the sink to the source, making the source uncommon. Once again, the proof that making such a transfer is reasonable is given at the end of the proof of the lemma.

- If the saddle has index 3 or greater, then either the Consecutive Rule applies, or else each arc out of the source gets $3/2$ alternations, which gives the source value at least $9/8$ and thus makes it uncommon.

To complete the argument for degree-2 sources, and thus complete the proof of the lemma we must show that in the cases in which we transferred value, each inoperable vertex transferring value retained enough value to stay uncommon. We start by noting that all transfers will be from sinks of degree 3 or higher to sources of degree 2, or (in the symmetrical argument for sinks) from sources of degree 3 or higher to sinks of degree 2. Thus there will be no conflicting transfers. We will prove that the sink-to-source transfers meet the stated conditions; the argument for source-to-sink transfers will be symmetrical.

We start by noting that each such sink has a value of at least degree$(t)/2$. The sink must receive value of at least $1/2$ for each arc into a saddle (if the saddle has index 1 neither the Index-1 Rule nor the Consecutive Rule can apply to t, which is inoperable, so there is at most one adjacent eligible source at that saddle and t gets value of at least $1/2$; if the saddle has index 2 or greater the Consecutive Rule cannot apply, so t gets at least $3/2$ alternations, which has a value of $1/2$ or more).

Next we consider a sink t that transfers value to one or more sources. We observe that each arc a into t can be adjacent (in the cyclic order at the saddle at the a's tail) to at most one arc out of some source (if not, the Consecutive Rule would apply at that sink, which is inoperable). Since each source that receives value from t is incident to two arcs each of which is adjacent to a distinct arc into t, the number of such sources can be at most $\lfloor \text{degree}(t)/2 \rfloor$. If we transfer $1/4$ to each such source, t's remaining value is at least

$$\frac{\text{degree}(t)}{2} - \frac{\text{degree}(t)}{2} \cdot \frac{1}{4} = \frac{3 \cdot \text{degree}(t)}{8},$$

which is greater than or equal to $9/8$ for sinks with degree 3 or greater. Thus the sinks that transfer value will remain uncommon.

This completes the proof of the lemma. □

The preceding lemma dealt with sources and sinks that have been cleaned up. However, the cleanup algorithm is not applied to all sources and sinks. The next lemma will show that there are not too many sources and sinks that haven't been cleaned and that are not adjacent to an operable arc. This, along with the fact that there aren't too many uncommon sources and sinks, will allow us to argue that the number of operable arcs is suitably large. We start with some definitions and observations.

A **problem high-degree source** is a source of degree greater than the constant d (introduced in Section 3.3) with all arcs out either unique-in or locally unique-in. A **problem high-degree sink** is a sink of degree greater than d with all arcs in either unique-out or locally unique-out. Such vertices are problems in the sense that they may have no operable arcs and they can't be cleaned up in constant time.

We define a **simplified underlying graph** of an embedded planar DAG $G = (V, A)$ to be the embedded planar graph $G'' = (V'', E'')$ that results when each set of parallel edges in G' (the underlying graph of G) is replaced by a single edge. G'' has the following properties:

- Any problem high-degree source or sink in G has no parallel arcs, and hence will have the same degree in G'' as it has in G.
- All faces in G'' have boundaries of length 3 or longer because there are no loops and because faces in G' with boundaries of length 2 are formed by parallel edges. Thus Inequality (2) in Section 2.1 holds.
- The number of vertices is the same in G'' and in G.

Given these facts, it's easy to prove the following lemma:

Lemma 4.7 *In any embedded planar DAG consistent with our invariants the number of problem high-degree sources and sinks is less than* $6n/d$*.*

Proof: Let l be the number of vertices in G'' that have degree greater than d. Because Inequality (2) from Section 2.1 holds for G'', we have

$$\frac{d \cdot l}{2} < |E''| < 3n \rightarrow l < \frac{6}{d} \cdot n.$$

Since every problem high-degree source or sink in G has degree greater than d in G'', the number of such sources and sinks must also be less than $6n/d$. $\square$

We now set $d = 1512$, which by the argument in the preceding lemma implies that the number of problem high-degree sources and sinks in the graph will be less than $n/252$.

Lemma 4.8 *In any cleaned-up embedded planar DAG G consistent with our invariants in which the number of sources and sinks is greater than or equal to* $n/14$*, a constant fraction of the arcs are operable.*

Proof:

As in previous proofs, let k be the number of sources and sinks.

We start with some preliminary observations: We can partition the arcs into two sets: those that have another parallel arc and those that don't. Since G has no cycles, every arc parallel to some other arc is both T and B with respect to the face common to the two arcs, and is hence operable by the TB Rule. From our discussion above about the simplified underlying graph G'', the number of arcs in G without parallels plus the number of sets of mutually parallel arcs is less than $3n$ (this number is $|E''| < 3n$).

Our goal is to specify a set of operable arcs such that the size of this set is a constant fraction of the number of sources and sinks in the graph (i.e., a constant fraction of k). The operable arcs we specify will correspond to edges in G'': they will be either arcs without parallels or single representatives of sets of parallel edges. Because k is at least a constant fraction of n by the conditions of the lemma, and because the arcs in G that don't correspond to unique edges in G'' are all parallel arcs (and thus operable), specifying this set of operable edges will imply that a constant fraction of the arcs in G are operable.

We now specify the set of operable arcs in the following way: For each source(sink) at the tail(head) of at least one TB arc, put one such arc in the set; the arc is operable by one of the TB Rules. (To be consistent with the condition that we choose only one arc corresponding to any edge in G", if the arcs specified for a source and a sink both come from the same parallel set, then a single arc representing the parallel set will be used for both the source and sink.) If the source or sink has degree 1, then the incident arc is operable by the Degree-1 Rule and is added to the operable set. The remaining sources all have only unique-in and locally unique-in arcs out; the remaining sinks only have unique-out and locally unique-out arcs in. We will ignore problem high-degree sources and sinks for the moment, so we can assume that all edges out of sources and into sinks are clean. Thus, for any other source incident to a unique-in arc or sink incident to a unique-out arc we can add such an arc to the set of operable arcs because of the Unique-In/Unique-Out Arc Contraction Rule. This leaves only sources with clean locally unique-in arcs out and sinks with clean locally unique-out arcs in; by Lemma 4.6 every such source or sink is either uncommon or at the tail or head respectively of an operable arc.

An operable arc has been specified for every source or sink that is not either a high degree problem or uncommon. The number of problem high-degree sources and sinks is less than $n/252$ by Lemma 4.7 and the choice of d; this is less than $k/18$ by the conditions of this lemma. The number of uncommon sources and sinks is less than $8k/9$ by Lemma 4.5. Thus the number of sources and sinks for which an operable arc has not been specified is less than $17k/18$, which means that an operable arc has been specified for more than $1/18$ of the sources and sinks.

In order to complete the proof, we must show that we don't have too many duplicate arcs in the set. Since every operable arc we've specified is incident to a source or a sink, we've only included duplicates when an arc in the set was specified for both a source and a sink. In that case we could include an arc at most twice; thus, the size of the set is at least a constant fraction of k. $\square$

5 Conflict Resolution

In the previous section we showed that in any embedded connected planar DAG meeting certain invariants a constant proportion of the arcs are operable once the graph has been cleaned up. However, applying the reduction rules to these operable arcs leads to two types of conflicts we must deal with: **intra-rule conflict**, and **inter-rule conflict**.

Intra-rule conflict arises when we try to apply in parallel a single rule to all arcs operable by that rule. Doing so can lead to cases in which either invariants aren't maintained or in which the rule applications cannot be completed in some specified amount of time (for DAG reduction, this will be constant time; in the general reduction algorithm this will be $O(\log n)$). For example, removing two arcs, both of which are marked T and B, can result in a graph that doesn't meet the invariant that all faces are flow faces (see Figure 16 below). Another potential problem is that

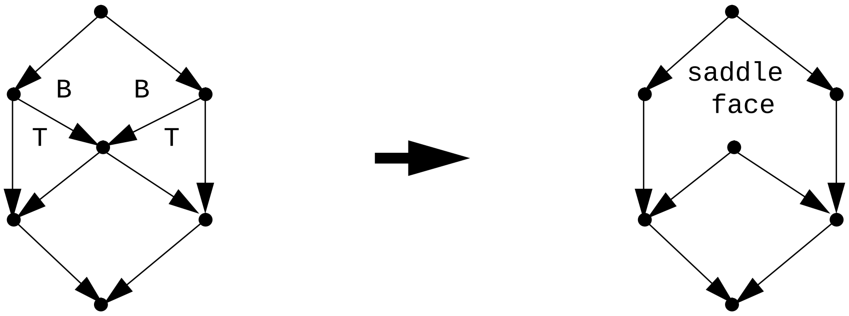


Figure 16: Example of intra-rule conflict for TB Rule

removing multiple arcs via the TB Rule could leave us with an arbitrary number of arcs that must be combined into a single topological arc (see Figure 17 below). Updating the information for all the internal components (e.g., determining leader information and rank ordering) could thus take time $O(\log |A|)$. Likewise, it must be possible to combine faces in constant time (since we don't

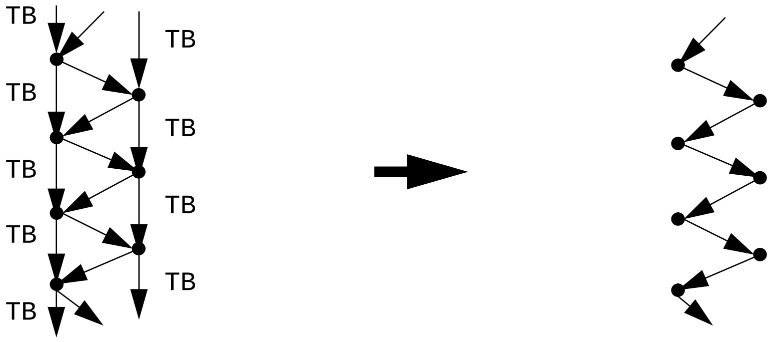


Figure 17: Creation of topological arc with arbitrarily many segments

keep rank orders on face boundaries, it is possible to combine an unbounded number of faces into a single face, but we must show that this does not require excessive time).

Inter-rule conflict arises when applications of a particular rule make arcs that were operable by another rule inoperable. Both types of conflict affect our counting argument aimed at showing a constant proportion of the arcs are removed in each pass through the main loop.

Before we discuss conflicts and conflict resolution in detail, we recall the following observation made in Section 2.2, which is useful in a number of subsequent arguments: *A flow face has at most one source and one sink on its boundary.*

5.1 Resolution of Intra-Rule Conflict

We will deal with conflicts between applications of a single rule by building a **conflict graph** that relates the conflicting arcs. The graph consists of a vertex for each arc operable by the rule in question, and an edge between each pair of these vertices where the removal of the corresponding arcs causes a conflict. The edges can be undirected or directed, depending on whether the conflicts are symmetric or asymmetric. It is clear that choosing an independent set from the conflict graph will give a set of arcs that don't conflict with each other. We will show how to find an independent set that includes at least a constant proportion of the vertices in the conflict graph, and thus a constant proportion of the arcs operable by a particular rule. In general, the conflict graphs have bounded degree, so finding a maximal independent set (MIS) in the conflict graph will suffice.

The intra-rule conflict definitions for each rule follow. In all cases but one we state the maximum degree of the conflict graph. We argue that the "flow faces only" invariant is preserved. We also argue that any changes resulting from removing non-conflicting arcs can be processed in constant time in the CRCW model; in particular, we show that we never have to combine arbitrarily many arcs into a single topological arc, and that we never have to splice the arcs from arbitrarily many vertices into consecutive places in the cyclic order at some vertex. Also, we must show that where rules combine faces, the associated work can be done in constant time. The rule-by-rule conflict definitions are as follows:

[TB Rule] For the TB Rule we break the conflict resolution into four stages. In each stage we determine conflicts for a particular type of TB arc, then remove non-conflicting arcs of that type. For purposes of the counting argument, we assume that we first determine all TB arcs, then apply the conflict resolution procedure. At the time that conflict resolution for a particular type of TB arc occurs, arcs of that type are specified. The reason for this is that as TB arcs are removed, the formation of topological arcs can change the characteristics of a particular arc. For example, an arc meeting the conditions for Type II TB arcs prior to arc removal could meet the conditions Type III TB arcs after the removal of a Type I TB arc. The four types are as follows:

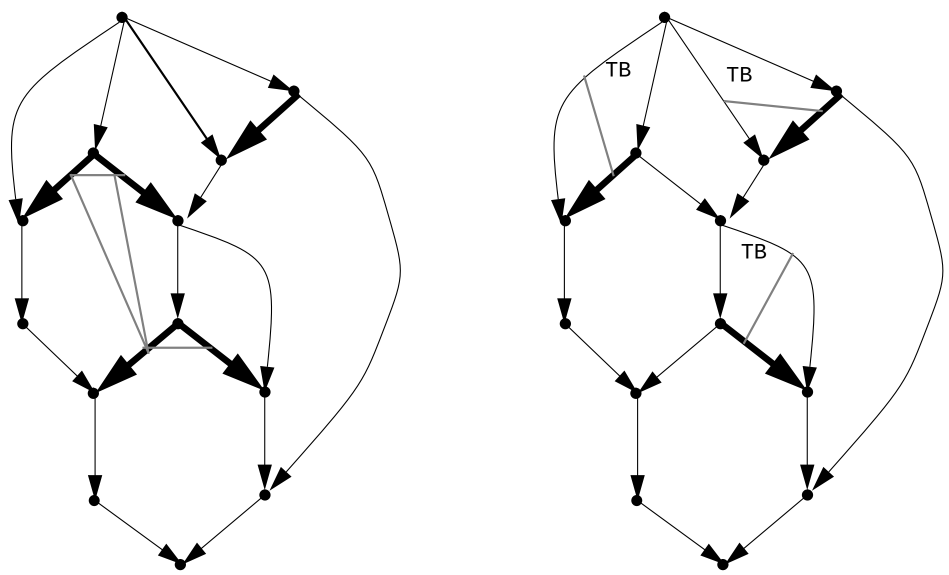
- A **Type I** TB arc is not marked both T and B for any single face.
- A **Type II** TB arc is marked both T and B for exactly one face f, and the other T and B marks for f are not common to a single arc.
- A **Type III** TB arc is marked both T and B for exactly one face f, and the other T and B marks for f are common to a single arc.
- A **Type IV** TB arc is any TB arc that is marked both T and B for two faces.

We describe the conflict resolution for each type of arc in turn:

> **[Type I TB Arcs]** A Type I TB arc a conflicts with any other Type I TB arc that is marked T or B with respect to a face for which a is marked T or B. Each Type I arc conflicts with at most 6 other arcs (an arc can lie on two different faces and there are up to 3 arcs on each face with which it will conflict); these conflicts are symmetrical (an example of a conflict graph

is shown in Figure 18 below). A MIS is selected from the conflict graph and the associated arcs are selected for removal.

Note that removal of these Type I arcs might either make certain Type II and Type IV TB arcs inoperable, or might make them into Type I arcs that won't get removed during this arc-removal phase. For this to happen, however, these arcs must be marked T, B, or both T and B on a face from which a Type I TB arc is removed. Thus vertices corresponding to these arcs could be added to the Type I conflict graph without increasing its maximum degree and without changing the fact that the MIS is maximal. (This extension of the conflict graph is not necessary in the actual algorithm; it is a counting mechanism that will be used in the proof that the conflict algorithm will allow us to remove a sufficient number of arcs. The fact that a Type II or Type III arc has become inoperable can be detected in a subsequent arc removal step.) An example of how Type II TB arcs can be added to the conflict graph is shown in Figure 18.



Conflict Graph for Type I TB Arcs: Heavy Arcs are Type I; Light Lines Indicate Conflicts

Results for Type I TB Arcs: Heavy Arcs are Type I MIS; Light Lines Indicate Type II Conflicts

Figure 18: TB Rule Conflict Graphs

To see that this conflict resolution procedure will preserve the "flow faces only" invariant, note that saddles will result only if the removal of some set of arcs effectively changes the marks on some other removed arc so that it is no longer marked both T and B. To create such a conflict with a Type I TB arc a, we must remove an arc a' that is common to a face f with a, and that has the same mark as a with respect to f. Since we only remove Type I TB arcs at this time, such an arc must be Type I, and our conflict procedure rules this out.

The TB Rule can never create cycle faces in a DAG, since it only removes arcs.

The following lemma shows that this procedure combines at most a constant number of arcs into a topological arc:

Lemma 5.1 *At most 3 arcs can be combined into a single topological arc as a result of removing Type I TB arcs.*

Proof: We number the arcs that are incorporated into a new topological arc according to their order of occurrence on this new arc, with the arc closest to the tail being numbered 1. We will refer to the i^{th} arc as a_i. Suppose v is the tail of some a_i, and that v becomes internal to the new topological arc. Then every arc incident to v other than a_{i-1} and a_i must be removed at the same time. Note that if two such incident arcs were adjacent in the cyclic order, the conflict resolution procedure for Type I TB arcs would prevent the removal of one of them. Therefore if we consider the cyclic ordering at v in the clockwise direction, there can be at most one Type I TB arc between a_{i-1} and a_i, and at most one between a_i and a_{i-1}.

Now assume that removing Type I TB arcs can cause four or more arcs to be combined into a single new topological arc (the following argument is illustrated in Figure 19 below). This implies that at least one arc incident to v_4, the tail of a_4, must be removed. We will refer this arc as b_4. Without loss of generality, assume that b_4 lies between a_3 and a_4 in the clockwise cyclic order at v_4. b_4 and a_3 are common to a face we will refer to as f_1.

Another arc b_3 incident to v_3, the tail of a_3, must also be removed. If b_3 lies between a_2 and a_3 in the clockwise cyclic order at v_3, then it also is common to f_1, and conflict resolution will prevent the simultaneous removal of b_3 and b_4. Therefore b_3 must lie between a_3 and a_2 in the clockwise cyclic order at v_3. b_3 and a_2 are common to a face f_2.

Finally, a Type I TB arc incident to v_2, the tail of a_2, must also be removed. However, if this arc lies between a_2 and a_1 in the clockwise cyclic order at v_2, it is common to face f_2 and conflicts with b_3; if it is lies between a_1 and a_2 in the clockwise cyclic order at v_2, it is common to face f_1 and conflicts with b_4 because there is no arc between a_2 and a_3 in the clockwise cyclic order at v_3. Thus our assumption leads to a contradiction, and the lemma is proved. □

To see that at most two crosspointers get spliced into one, first note that if more than two pointers are spliced together, then for any pointer other than the first or last, the arcs at both its head and tail must be removed. This requires the removal of two arcs from one face, which is prevented by the conflict resolution procedure.

Since we never remove more than one arc from any face, it is obvious that we never remove two arcs that are consecutive in the cyclic order at any vertex. Likewise, we never combine more than two faces into one.

[Type II TB Arc] The conflict resolution procedure for Type II TB arcs is different from the procedure for most other rules because it involves the construction of two conflict graphs in sequence, and it differs from all other rules because the first conflict graph is a forest and may not be of bounded degree. The vertices of the first conflict graph represent Type II TB

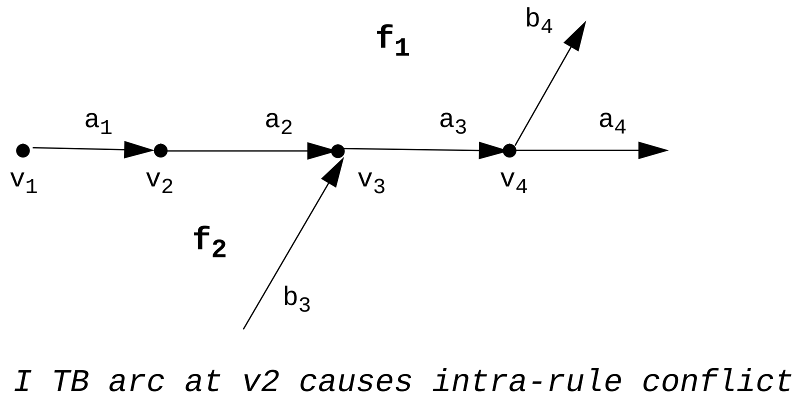


Figure 19: Example for Topological Arc Formation

arcs that remain operable after conflict resolution for Type I TB arcs is completed. (Recall that we are considering arcs that were operable prior to the removal of any Type I TB arcs; any newly-created Type II TB arcs are considered inoperable, as are those made inoperable by Type I conflict resolution. Also note that we are only considering arcs that are currently Type II; any operable Type II arcs that became Type III when Type I arcs were removed will be considered later.) It is easy to see that any such arc meets two easily-tested conditions: it was operable prior to Type I removal, and it currently meets the conditions for Type II TB arcs. Each such Type II arc a is on the boundary of a unique face f for which a is not both T and B. If the opposite side of f is an operable Type II arc b, direct an arc in the conflict graph from the vertex representing a to the vertex representing b. The result is a directed forest (see Figure 20). Removing Type II TB arcs does not affect the operability of Type III or Type IV TB arcs.

We could use tree contraction to find an independent set in the forest that contains at least half the vertices. However, that could take time $O(\log |A|)$. Therefore we use a simpler method that runs in constant time in the ARBITRARY CRCW model. To simplify the exposition, we first introduce some terminology: a **chain vertex** is any vertex in the conflict forest with indegree 1, provided that the arc in is not incident to a leaf. The first conflict resolution algorithm for Type II TB arcs can now be stated:

- Add all leaves in the forest to the set of arcs to be operated on.
- Form the subgraph induced by the chain vertices and replace the directed arcs by undirected arcs. This subgraph will have maximum degree 2. Find a MIS in the subgraph and add the corresponding arcs to the set of arcs to be removed.

The argument that this gives us at least $1/3$ of the operable Type II TB arcs is straightforward. We first note that since the subgraph induced by the chain vertices has maximum degree 2, the

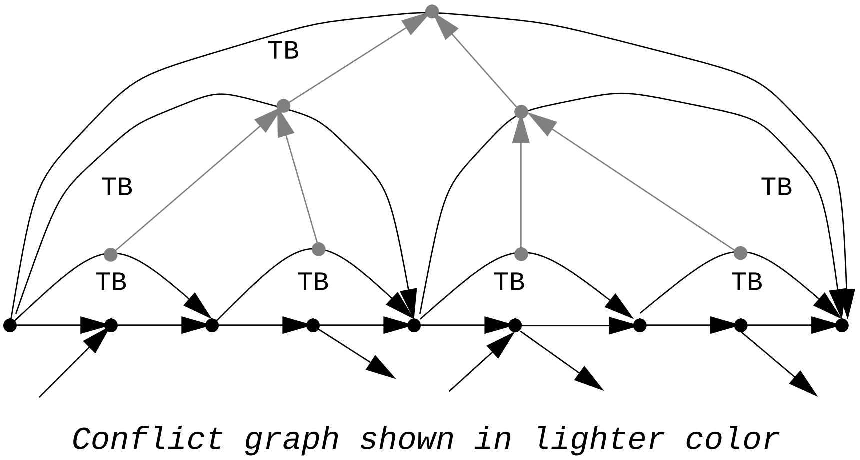


Figure 20: Conflict Graph for First Phase of Type II TB Conflict Resolution

MIS has at least $1/3$ of these vertices. Next we count the vertices other than chain vertices. There are three types:

1. leaves
2. vertices with indegree 2 or greater. We noted in the proof to Lemma 4.2 that the number of such vertices must be less than the number of leaves.
3. vertices with indegree 1 for which the arc in is incident to a leaf. The number of such vertices is at most the number of leaves.

Thus the number of non-chain vertices is less than three times the number of leaves. Since all arcs corresponding to leaves will be operated on, this is greater than $1/3$ of the remaining arcs, and the claim is proved.

The second phase of conflict resolution prevents the creation of topological arcs out of more than some constant number of arcs. The conflict rule is as follows: for each Type II TB arc a selected in the first round of conflict resolution, consider the other T and B arcs on the face f for which a is marked both T and B. Each lies on the boundary of another face (f_1 and f_2 respectively; they need not be distinct). If the T arc is marked either T or B on f_1, and if the boundary of f_1 opposite from the T arc is a Type II TB arc marked both T and B on f_1 and was chosen in the first round, put an edge between the vertices representing it and a in the conflict graph. Likewise, if the B arc is marked either T or B on f_2, and if the boundary of f_2 opposite from the B arc is a Type II TB arc marked both T and B on f_2 and was chosen in the first round, put an edge between the vertices representing it and a in the conflict graph. These conflicts are symmetrical, so the degree of any vertex in the conflict graph is at most 2, and a MIS from this graph will correspond to a set of arcs that is at least a constant fraction of the Type II TB arcs that were operable at the start of this conflict resolution stage.

To see that the two-step conflict resolution procedure will preserve the "flow faces only" invariant, note that saddles will result only if the removal of some set of arcs effectively changes the marks on some other removed arc a so that a is no longer marked both T and B. If a is a Type II TB arc, this would require the removal of at least one arc marked either T or B on the face for which a is marked both T and B. Since we are only removing Type II arcs at this stage, this is ruled out by the first conflict rule.

Removal of the arcs selected by this process will cause at most three consecutive crosspointers to be spliced into one. If more than two pointers are spliced, then some pointer p must lie between two arcs that are removed; these arcs must be common to a single face. Our reduction rules will insure that if two arcs are removed from a single face then neither is marked both T and B on that common face (if both are, they are not Type II TB arcs; if one is then the two arcs conflict in the first half of the conflict resolution step). Therefore, any pointer spliced to p must cross a face to or from a the side of a Type II TB arc that is marked both T and B. But the Type II TB arc is the only one removed from such a face, and the chain of splices can't extend any further. Thus at most three pointers get spliced into one (one to a Type II arc, one from one Type II arc to a second, and one from the second Type II arc).

The same sort of argument shows that the maximum number of arcs consecutive in the cyclic order that are removed at some vertex is at most two. Any pair of such arcs are common to a face f. If the arcs have the same orientation (i.e., both are in-arcs), then by the argument above neither is both T and B on the common face. If they have opposite orientations, then neither can be both T and B on the common face. Therefore in either case they must be marked both T and B on the other faces; no other arcs are removed from these faces, so the next arcs in the cyclic order in either direction will not be removed.

We still need to show that the number of arcs that are combined into a single topological arc is bounded by a constant. We have just shown that the algorithm will never remove three consecutive arcs in the cyclic order at any vertex. We can apply this fact at any v that becomes internal to a topological arc, so we only need to consider the cases in which one or two consecutive arcs are removed. If two consecutive arcs are removed, we can limit the possibilities to four as shown in Figure 21 below. The unlabeled arcs are components of the topological arc. In (a) both $a1$ and $a2$ could be Type II TB arcs and could both be removed without conflict. In (b) and (c), there is exactly one way in which both arcs could be Type II TB arcs; in those cases the arcs will conflict by the first conflict resolution step. In (d) there is no way that the two labeled arcs could both be Type II TB arcs.

Now consider the possible configurations of arcs incident to a vertex v that will become internal to a topological arc. At least one arc incident to v must be removed. The configuration of arcs at one side of v (i.e., either clockwise in the cyclic order between the arc into v and the arc out of v, or clockwise in the cyclic order between the arc out of v and the arc into v) can be one of four things: no arcs, an arc in, an arc out, or two arcs as specified above. We note that if there is an in arc on each side of the topological arc at v then these two arcs conflict by the second conflict resolution step; this is also the case if there is an arc out on each side. Therefore we are left with the four possibilities (plus their mirror images) shown in Figure 22 below.

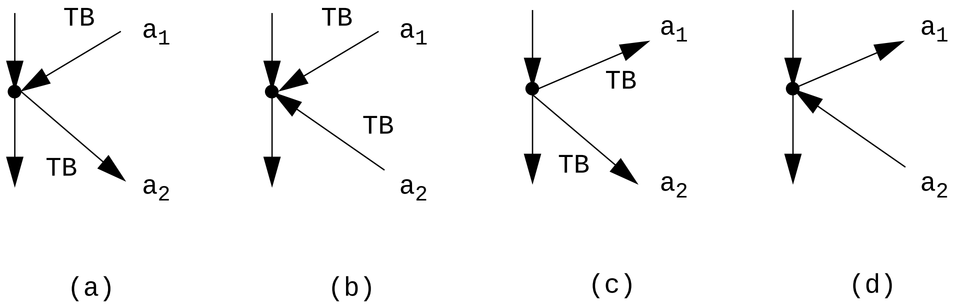


Figure 21: Consecutive Type II TB Arc Incidences at a Vertex

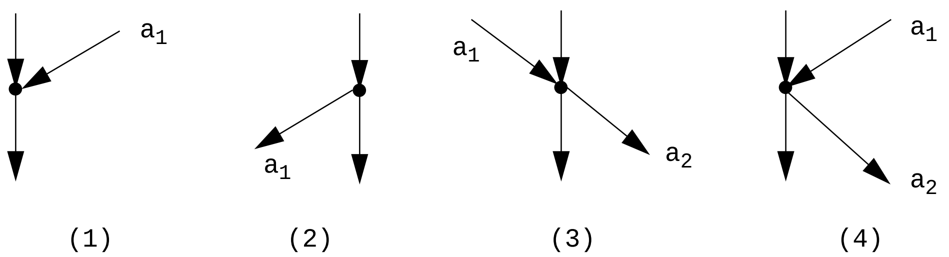


Figure 22: Nonconflicting Type II TB Arc incidences at vertices that could become internal

Finally consider the configurations possible at two consecutive vertices u and v that become internal to a topological arc. Specifically, let there be an arc from u to v that becomes part of the topological arc, and let the arc out of v be the last arc in the sequence that becomes the new topological arc. First consider the case in which there is an in-arc a_{in} incident to v. This in-arc will conflict with an out-arc on the opposite side of u, and an out-arc at u on the same side of the topological arc as a_{in} cannot be a Type II TB arc. If there is no out-arc at u, there must be an in-arc at u. However, such an arc will conflict with a_{in} if it lies on the same side of the topological arc. Thus, if configurations (1), (3), or (4) shown in Figure 22, or their mirror images, occur at v, there is only one allowable configuration that can occur at u: configuration (1), with the arc on the opposite side of the topological arc from a_{in}. By the same argument, the only configuration that can occur at the vertex preceding u on the topological arc is configuration (1), with the arc on the same side of the topological arc as a_{in}. But such an arc must conflict with a_{in} by the first conflict resolution rule. Thus in this case at most three arcs can be combined into a single topological arc.

Now consider the case in which the configuration at v is (2). If configuration (1), (3), or (4) occurs at u, the previous argument says that at most three arcs preceding the arc out of v will be combined into the topological arc, limiting the total number of arcs combined to four. If configuration (2) occurs at u, the arc out of u must be on the opposite side of the topological arc from the arc out of v; if not, they'd conflict by the first conflict resolution rule. This implies that configuration (2) can't occur at the vertex preceding u; if one of the other configurations does occur at the preceding vertex, we again have the case discussed above, and at most three more arcs can be included in the new topological arc. Thus, at most five arcs can be combined into a single arc as a result of removing Type II TB arcs.

Type II TB arc removal is one case where an unbounded number of faces can be combined into one. Let f be a face such that an arc marked both T and B with respect to f is removed. The conflict resolution rules will prevent f from being combined with another such face. However, several faces such as f can be combined with a face f' such that no Type II TB arc marked both T and B with respect to f' is removed. If this occurs, the processor for f' will become the processor for the new face. To create this new face, f' needs to determine if the leaders have changed as a result of the removal of a Type II TB arc. Then the various faces to be combined with f' need to set the processor of f' as the new processor. The edges on the remaining boundaries of these faces can read the new processor number to complete the change. This can all be done in constant time in the CRCW model, so combining an arbitrary number of faces in this way is not a problem.

[Type III TB Arc] Recall that a Type III TB arc is marked both T and B for exactly one face f, and the other T and B marks for f are common to a single arc b. We apply the conflict rules for Type III TB arcs to any such arc that is currently operable. Such an arc a meets the following two conditions: a was marked both T and B prior to the start of TB arc conflict resolution, and a currently meets the conditions for Type III TB arcs.

The first conflict rule says that if b is also an operable Type III TB arc, a conflicts with b and vice versa.

In addition, we need some conflict rules to prevent the formation of topological arcs from arbitrarily many arcs. To understand these rules, it is first useful to discuss the situations in which vertices can become internal as a result of Type III TB arc removal.

The first conflict rule will assure that if Type III TB arc a is removed, then an arc parallel parallel to a will remain in the graph. This implies that the alternation number of any vertex in the graph will not decrease as a result of Type III TB arc removal. Thus only flow vertices can become internal to topological arcs.

Also, for each arc b remaining after Type III TB removal, at most two Type III arcs parallel to b could have been removed. Thus if the outdegree (respectively indegree) of a flow vertex is four or more, that vertex cannot become internal as a result of the removal of Type III arcs.

The additional conflict rules will thus apply to operable Type III TB arcs that are incident to flow vertices that have indegree and outdegree less than or equal to 3. Let a be such an arc and v be such a flow vertex at the tail of a. Once again, b will denote the TB arc common to the face for which a is marked both T and B.

To determine these additional conflicts, consider the next arc in the cyclic order at v, where the direction of the order is determined by b followed by a. If c is out of v, then a has no conflicts with respect to its tail. If c is into v, the following cases apply:

- If c is a Type III TB arc, the following conflicts can occur: a and c conflict if c is operable; if the arc d opposite c on c's TB side is an operable Type III TB arc, then a also conflicts with d (this last condition applies whether or not c is operable).
- If c is not a Type III TB arc, then we consider the indegree of v. If the indegree is greater than one, a has no more conflicts with respect to v. If c is the only arc in, then we consider the vertex u at the tail of c (if the indegree of v is greater than one in this case, v will not become internal). If u is a flow vertex with indegree 3 or less and c is the only arc out, then we consider the arcs into u (if u does not meet these conditions, u will not become an internal vertex). Let d be the arc into u on the face common with a. If d is a Type III TB arc, the following conflicts can occur: a and d conflict if d is operable; if the arc e opposite d on d's TB side is an operable Type III TB arc, then a also conflicts with e (this last condition applies whether or not d is operable). If d is not a Type III TB arc, no conflicts occur.

These conflicts are illustrated in Figure 23 below. A symmetrical set of conflicts occur for arcs out of the head of a.

As in previous cases we build a conflict graph with vertices corresponding to the Type III TB arcs, and edges corresponding to conflicts. In this case conflicts may not be symmetrical; however, for any operable Type III TB arc there are at most five arcs with which it can conflict and that can conflict with it. The degree of any vertex in the conflict graph is at most five. Thus we can get a constant proportion of the vertices in the conflict graph by finding a MIS. As was the case for Type I arcs, removal of Type III arcs can affect the operability of other types of TB arcs. In particular, certain Type IV TB arcs can either become inoperable, or can become Type III TB arcs that won't be removed during this phase of arc removal. However,

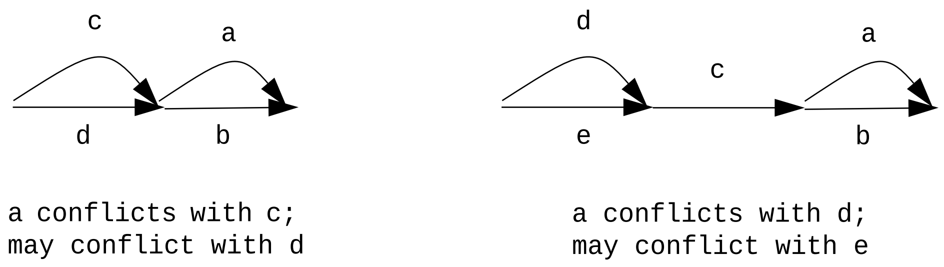


Figure 23: Type III TB Arc Conflicts

all such Type IV arcs will be common to a face for which a removed Type III arc is marked both T and B. As was the case for Type I conflict resolution, we can extend the conflict graph to include these arcs after we've selected a MIS: If a is represented by a vertex in the MIS in the conflict graph, and if a is labeled both T and B on a face f, and the opposite side of f is a Type IV TB arc b, add a corresponding vertex and edge to the conflict graph (again, this is done for counting purposes of the proof and need not be done in the actual algorithm). This doesn't increase the maximum degree of the conflict graph, and that, since every added edge is incident to an element of the MIS, the MIS is still a MIS.

To see that this will preserve the "flow faces only" invariant, note that saddles will result only if the removal of some set of arcs would effectively change the marks on some other removed arc a so that a would no longer be marked both T and B. This can only occur for Type III TB arcs if we remove two arcs both of which are marked both T and B on a common face. This is ruled out by the conflict rules.

Removal of the arcs selected by this process will cause at most three consecutive crosspointers to be spliced into one. If more than two pointers are spliced, then some pointer must lie between two arcs that are removed; these arcs must be common to a single face. Our conflict rules will insure that if two arcs are removed from a single face then neither is marked both T and B on that common face (if either is, then by the definition of Type III TB arcs both are, and they conflict). Therefore, any pointer spliced to such a pointer must cross a face to or from a the side of a Type III TB arc that is marked both T and B. But the Type III TB arc is the only one removed from such a face, and the chain of splices can't extend any further. Thus at most three pointers get spliced into one (one to a Type III arc, one from one Type III arc to a second, and one from the second Type III arc).

It is easy to see that no more than two arcs consecutive in the cyclic order at any vertex are removed. The conflict rules insure that if a Type III arc a is removed, then the other arc on the face for which a is marked both T and B is not removed. If two consecutive Type III arcs are removed, then neither can be marked both T and B on the face they share. Further, the next arc in either direction in the cyclic order will remain.

To see that no more than three arcs are combined into a single topological arc, first recall that if a vertex becomes internal, either its outdegree, its indegree, or both are reduced to one (either the indegree or the outdegree could be one already). Because of the first conflict rule, if the outdegree (respectively indegree) is reduced, we can conclude that it was two or three prior to removal, and that the removed arcs were parallel to the remaining arc. It is easy to see that if a vertex v has both indegree and outdegree of two or three, then the additional conflict rules will insure that v does not become internal (an example of such a vertex is included in Figure 24 below as the "excluded" case; the arcs labeled a and b conflict since the other two arcs must both be Type III TB arcs even if they are no longer operable).

This allows us to reduce the number of cases we need to consider. There are four basic cases as shown in Figure 24 below (cases 1 and 3 each represent two mirror-symmetric configurations). Assume that v is the last vertex that will become internal in the newly formed

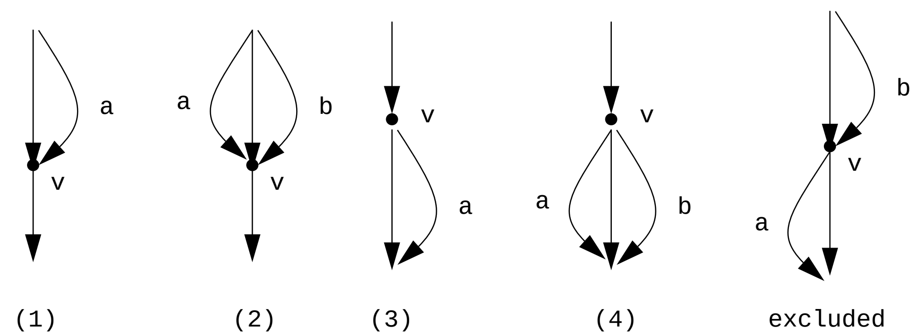


Figure 24: Cases for showing that Type III TB conflict resolution prevents the formation of long topological arcs

topological arc. Let a_{top} be the arc with v as its head that is included in the topological arc, and let u be the tail of a_{top}. If u also could become internal and the configuration at v was either 3 or 4, then the configuration at u would have to be 1 or 2 respectively (operable arcs out of u would be arcs into v, which are not consistent with the assumed configuration). In either case conflicts occur and not all the arcs incident to u and v will be removed (note that for configurations 1 and 3, the arc parallel to the Type III TB arc must also be a Type III arc). If the configuration at v is 1 or 2, then either configuration 3 or 4 occurs at u (the arcs out of u are the arcs into v); in this case its clear that at the next vertex back from u on the desired topological arc there will be conflicts as described above. In this case it is possible to combine three topological arcs into a single arc.

Removal of Type III TB arcs can lead to the combination of an unbounded number of faces in the same way as for Type II TB arc removal. The method of combination and the argument that it will take constant time are the same as in the Type II case above.

[Type IV TB Arc] Recall that a Type IV TB arc is marked both T and B for two faces. The Type IV arcs that are removable at this step meet the easily-tested conditions that they were marked both T and B prior to TB arc removal, and that they currently meet the conditions for Type IV TB arcs.

Each Type IV TB arc a lies on the boundary of two faces. If a is operable, it will conflict with any operable Type IV TB arc that forms the opposite side of either of these faces. Thus each such arc can conflict with at most two other arcs, and all conflicts are symmetrical. The conflict graph has a maximum degree of 2.

It is easy to see that two arcs consecutive in the cyclic order are never removed: if the opposite sides of the faces bounded by a removed arc are not Type IV arcs, they are unaffected; if they are Type IV arcs they will conflict with the removed arc and not be removed. Thus at most one arc is removed from any face. When a single Type IV TB arc between two faces is removed, the two faces are merged into a single flowface, so no saddle faces are formed. Also, at most two pointers can be spliced into one; if longer pointer chains were formed, two arcs would have to be removed from some face, which is not possible.

It is also clear that removal of a Type IV arc cannot create a topological arc. In general the conflict rules assure that both the top and bottom arcs on two faces stay incident to the head and tail of the removed arc. In the degenerate case in which the graph consists of two parallel arcs, the arc that remains is not combined with any other arc.

The conflict procedure for Type IV TB arcs prevents the removal of more than one such arc from any face. Therefore at most two faces are combined. This can easily be done in constant time.

This completes the conflict rules for various types of TB arcs. We can now argue that this four-step procedure is sufficient to remove a constant fraction of the TB arcs: We note that the maximum degree of the conflict graph for Type I TB arcs is 6, so the conflict resolution procedure discussed in Section 5.1 above will yield a MIS that includes at least $1/7$ of the set that includes (1) arcs operable by this rule and (2) Type II and Type IV TB arcs that will not be removed as a result of the removal of Type I arcs corresponding to vertices in the MIS. The arcs not in this set are Type II, Type III, and Type IV TB arcs that remain operable after Type I removal.

The conflict resolution procedure for Type II TB arcs removes a constant proportion of the remaining operable Type II arcs without affecting the operability of the remaining Type III and Type IV TB arcs.

The conflict resolution procedure for Type III TB arcs removes a constant proportion of the remaining operable arcs that are either (1) Type III or (2) Type IV arcs that won't be operated on because of the removal of Type III arcs. The argument is as for the Type I case.

Finally, the Type IV conflict resolution procedure allows the removal of a constant fraction of the remaining arcs. Thus a constant proportion of the arcs that were TB arcs prior to conflict resolution will be removed.

[Degree-1 Rule] There are no conflicts between arcs operable by this rule, so the conflict graph has no edges. To see that no saddle or cycle faces are created, note that the removal of such an arc a changes only the face of which a is on the boundary. Furthermore, removing a degree-1 vertex and its arc causes the number of face alternations either to stay the same or to decrease. Thus, given that all faces are flow faces, we start with two alternations on the face. If the number of alternations goes down it must go to zero (there are always an even number of face alternations on a face), which would mean the face is a cycle face. But that implies that the remaining boundary formed a cycle in the original graph, which contradicts the fact that the graph is a DAG.

Given that the graph being reduced contains no directed cycles and that there is at most one source and one sink on a flow face, it is easy to show that there are at most two arcs operable by this rule on any face and these arcs can't be adjacent in the cyclic order at any vertex. There are no problems with processing time (i.e., with respect to pointer splicing or updating topological arc information) in this case.

The remaining rules affect only clean arcs. Therefore we don't need to worry about splicing pointers: A clean arc has no pointers through internal vertices, so its removal won't cause any splicing. If it is contracted, the head will become top (respectively, the tail will become bottom), and any incident pointers can be deleted (the conditions on clean arcs insure they won't become self-loops or backpointers). However, we do need to be careful that no more than a constant number of faces get combined into a single face as the result of the application of some rule.

[Unique-In(Unique-Out) Arc Contraction Rule] An arc a operable by this rule conflicts with its two neighbors in the cyclic order at the source(sink). The conflicts are symmetrical, so the degree of any vertex in the conflict graph is at most 2. Contraction of a unique-in(unique-out) arc won't create a cycle or saddle face: Only the two faces that have this arc on a boundary are affected. Since the next arcs in a traversal of these faces have the same orientation on the boundary as the contracted arc, the face remains a flow face, and the "all flow faces" invariant holds. The conflict rule insures that consecutive arcs in the cyclic order won't be contracted, so changes in the cyclic order can be processed in constant time in the CRCW model. There are no problems with either combining too many arcs into a topological arc or combining too many faces together: no topological arcs can be formed and no faces can be combined by this rule.

(Note that there are no conflicts in which this rule applies to an arc at both a source and a sink; if a unique-out arc from a source is the unique arc into a sink, then these vertices plus the arc form a complete connected component. Since the reduction rules won't disconnect a DAG this situation won't arise except in the case that the graph has exactly one arc.)

[Adjacent Degree-2 Sources and Sinks Rule] A conflict graph can be constructed as follows: Vertices in the conflict graph will be the operable degree-2 sources. Each such source s checks for each sink t with which it is operable (there are at most 2) if there is a second source s' that is operable with t, and if so adds an edge to s' source in the conflict graph.

In addition, s may lie on a face f that has a degree-2 sink t as its bottom, and s is not operable with t. However, t may be operable with another source s'; if this is the case, then s and s' conflict. This conflict is defined in a symmetrical way with respect to s': if s' is operable with with a sink t, and if t lies on a face f such that the top of f is a source s that is operable, but not with t, then s' and s conflict. These conflicts are included to prevent the combination of more than a constant number of faces or the creation of a topological arc from arbitrarily many arcs.

Since the conflicts are symmetric, the maximum degree of any vertex in the conflict graph is 2. If any source selected for removal during conflict resolution is operable with more than one sink, it chooses one of the sinks arbitrarily.

If only non-conflicting sources and their corresponding sinks are removed from a graph that contains only flow faces, the conflict rules insure that each removal affects only three faces: face f_1 for which the source is top and the sink is bottom; face f_2 for which the source is top and the sink is not on the boundary, and face f_3 for which the sink is bottom and the source is not on the boundary (these faces are easily identifiable in Figure 3.3 in Section 3.2). When the source and sink are removed, the remaining face consists of the paths from the top of f_2 to the two saddle vertices and and the paths from the two saddles to the bottom of f_3. This forms a new flow face; the "all flow faces" invariant continues to hold. The conflict rules also insure that no more than these three faces are combined into a single face.

It is obvious that no more than two consecutive arcs in the cyclic order at any vertex are removed at once. Arcs are removed in pairs, so if more than two were removed simultaneously there must be at least two conflicting sources.

It is straightforward to show that at most three arcs are combined into a single topological arc. To see this, assume that four arcs could be combined into a topological arc consistent with the conflict rules, and let u, v, and w be the vertices that become internal in the order from tail to head of the topological arc. Since w is not already internal, some incident arcs must be removed. Assume that the removed arcs lie on a particular side of the topological arc. The conflict rules will not allow all arcs to be removed from either v or u on the same side of the topological arc. Therefore all arcs must be removed from v and u on the other side of the topological arc, and at least one arc must be removed at each of those vertices. But this would involve removing conflicting arcs, which contradicts our assumption.

[Source-Sink-Source (s-t-s)/Sink-Source-Sink (t-s-t) Rule] To make the exposition simpler, we will refer to the source involved in a potential application of the t-s-t rule as the "operable source", and the sink involved in a potential s-t-s Rule application as the "operable sink".

To prevent problems such as removing an arbitrary number of consecutive arcs in the cyclic order at some vertex or combining an arbitrary number of faces, the algorithm will apply these two rules in sequence (we will assume the t-s-t Rule is applied first, though the order is not important). The specific procedure will be as follows: first, mark all sources and sinks that are operable by these rules. Apply the t-s-t Rule (there are no conflicts between operable sources). Test whether the sinks marked as operable remain operable, and apply the s-t-s Rule to those that do (again, there are no conflicts between operable sinks). We will define conflicts between sources and sinks, though no conflict graph need be constructed - this is another case where we use conflicts for counting purposes only. To understand these conflicts, note that applications of the t-s-t Rule could leave a neighboring operable sink inoperable either because the number of neighboring sources drops to one, or because that sink ends up with too high a degree. Thus, every operable sink will conflict with every operable sink with which it is common to a face. Since operable sources have degree 3 or less, the number of sinks that can become inoperable by a single s-t-s Rule application is clearly bounded.

For a particular source at which the t-s-t Rule applies there may be more than one way to apply the rule. The source can arbitrarily pick one of the ways; this isn't a conflict in the sense we use the term. The same holds for applying the s-t-s Rule at some sink.

Each merging of cyclic orders at a source (for the s-t-s Rule) or a sink (for the t-s-t Rule) occurs between two arcs that are not removed, so consecutive merges don't occur. Thus there is no problem if a particular source appears in multiple s-t-s Rule applications or if a sink appears in multiple t-s-t applications (note that a high-degree vertex may be created as mentioned in the discussion of this rule in Section 3.2).

To see that the remaining graph has only flow faces, note that the removal of the arc (or arcs) out of any source affects only the structure of the two faces it borders (the case for sinks is symmetric). These two faces are replaced by two new flow faces. The same observation makes it clear that no arbitrary set of faces will be merged into one.

It is obvious that no topological arcs are formed.

[Consecutive Rule] Let a be an arc that is a candidate for removal by this rule, and let the two faces of which a lies on the boundary be f_1 and f_2. Then a conflicts with any other arcs that lie on the boundaries of f_1 and f_2 that are operable by the Consecutive Rule. These conflicts are symmetric. The conflict graph in this case has maximum degree 6.

It is obvious that this rule prevents the removal of successive arcs in the cyclic order at a source, sink or saddle vertex, and that the cyclic order at any combined sources or sinks can be updated in constant time.

It is easy to see that the application of this rule cannot produce saddle or cycle faces. Any single application of this rule affects only the two faces of which the removed arc lies on the boundary. These two flow faces are reconfigured into two new flow faces; the rest of the graph is unaffected. The same argument shows that a Consecutive Rule application never combines more than a constant number of faces into a new face. Also, no topological arcs are produced since all removed arcs are incident only to saddle vertices and sources or sinks.

[Index-1 Saddle Rule] An arc a operable by this rule conflicts with at most its two neighbors in the cyclic order at the source(sink), provided that those neighbors are incident to different saddles. It also may conflict with two arcs adjacent in the cyclic order at the saddle if these arcs are incident to sinks(sources) at the saddle (i.e., when this rule is applied at a source there may be two sinks at the index-1 saddle that also are operable by this rule). The degree of any vertex in the conflict graph is at most 4 (the conflicts are symmetric).

This conflict rule insures that if an arc is contracted then none of the adjacent arcs in the cyclic order at either end are affected, so there is no problem with splicing cyclic orders. Applications of this rule that involve two sources(sinks) only contract arcs, so no topological arcs will be formed, and faces will not be combined. In this case, the argument that no saddle or cycle faces are created is the same as for the Contraction Rule above.

Applications that involve a single source or sink adjacent to a saddle affect at most three faces. The separation of the graph does not create any topological arcs, and the affected faces all remain flow faces, though their boundaries are changed. The cyclic orders at the affected vertices can be modified in constant time.

With the exception of the first step of conflict resolution for Type II TB arcs, the degree of each vertex in the conflict graphs for each reduction rule is bounded by a constant. The conflict graphs are not necessarily planar, however (e.g., it is easy to construct graphs for which the Type I TB arc conflict graph is not planar). They are easily constructed in constant time in the CRCW model.

It is obvious that a maximal independent set (MIS) of vertices from a conflict graph represents a set of vertices that can be removed in parallel without problems; it is also obvious that a MIS in a bounded-degree graph contains a constant fraction of the vertices. Therefore we can use the techniques developed by Goldberg, Plotkin, and Shannon [GPS87] to resolve conflicts in $O(\log^* n)$ time.

If we introduce randomization, the running time can be reduced to constant time for the CRCW model. In particular, we can use Luby's Monte Carlo Algorithm A (described in Reference [Lub86]) for finding a MIS in constant time.

5.2 Conflict Resolution Between Rules

We deal with the second type of conflict by proving the following lemma:

Lemma 5.2 *Given the order of rule application specified in the algorithm description, a single application of any reduction rule reduces the number of arcs operable by subsequent rules (excluding the arcs removed by this rule application) by at most a constant number.*

Proof: The proof is by examining all cases.

The TB Rule affects the Degree-1 Rule only either where it lengthens (by making topological or by adding a new topological segment) the arc incident to a degree-1 vertex, or where it creates a new degree-1 vertex. There is no reduction in the number of arcs operable by the Degree-1 Rule.

All of the other rules operate on clean unique-in or locally unique-in arcs incident to sources (respectively unique-out or locally unique-out arcs incident to sinks); the s-t-s/t-s-t Rule additionally requires some locally unique-in/locally unique-out arcs that are not necessarily clean. These arcs will not be removed by the TB Rule. Further, it is obvious that a unique-in arc out of a source will remain unique-in if other arcs incident to its head are removed. It is possible that such an arc could become part of a topological arc if all but one arc out of the head are removed by the TB Rule, however. Any TB arc removed will affect at most one unique-in arc at its tail. (The symmetric argument holds for unique-out arcs into sinks.)

Recall that locally unique-in (respectively locally unique-out) arcs are incident to saddle vertices. Thus a locally unique-in arc a will remain locally unique-in: An arc adjacent in the cyclic order at the head of a is marked both T and B only if the next arc in the cyclic order has the same orientation. However, at each step of TB arc removal, the conflict resolution procedure will not allow the removal of two arcs with the same orientation that are adjacent in the cyclic order at some vertex. Finally note that if a is clean it will remain clean. If a pointer into its head were to be created across a face of which it lies on the boundary, an in-arc adjacent at the head would have to be removed. But the existence of such an adjacent arc would contradict the fact that this arc is locally unique-in. The arguments for locally unique-out arcs are symmetric.

Thus the application of the TB Rule doesn't conflict with any arcs operable by subsequent rules.

For Degree-1 Rule conflicts with subsequent rules, we first note that removal of an arc by this rule will not make any clean arc dirty. Also note that the removal of such an arc changes the structure of only one face. Thus it is easy to see that at most one conflict can occur with the Adjacent Degree-2 Source and Sink Rule, and at most one with either the s-t-s or t-s-t Rule. Since a degree-1 arc is adjacent to at most one other source or sink at a saddle vertex, at most one application of the Consecutive Rule can be in conflict. A degree-1 arc can be adjacent to at most one index-1 saddle, so there can be at most three conflicts with the Index-1 Saddle Rule. It is obvious that any arc operable by the Unique-In/Unique-Out Arc Contraction Rule is unaffected by the removal of an arc by the Degree-1 Rule.

For the Unique-In/Unique-Out Arc Contraction Rule, two key observations are that contraction of such an arc will never change the face structure of the graph (e.g., top and bottom of every face stays the same), and that arcs incident to saddle vertices will never be contracted. It is therefore easy to see that arcs operable by the Adjacent Degree-2 Source and Sink Rule, the Consecutive Rule, and the Index-1 Rule will not be affected. Recall that the s-t-s/t-s-t Rule is only applied at sources or sinks that have either two or three locally unique-out (respectively locally unique-in) arcs; this plus the observation that the face structure is unchanged imply that there will be no Unique-In/Unique-Out Arc Contraction Rule conflicts with s-t-s/t-s-t Rule applications.

Next consider the Adjacent Degree-2 Source and Sink Rule. Since the arcs incident to the source and sink involved are all removed, we don't have a conflict with the Consecutive Rule applied with either the source or sink as the center. This leaves at most four conflicts: a sink adjacent to the source in the cyclic order at either saddle vertex, or a source adjacent in the cyclic order to the sink at either saddle. For the s-t-s/t-s-t Rule there is a conflict if the sink might be involved in a t-s-t Rule application and the source in an s-t-s application (recall from the lemma statement that if an arc is operable by both rules it isn't counted as a conflict). Since the s-t-s and t-s-t Rules apply to sources and sinks that share faces, and since the degree of the source and sink in question is 2, there is at most one conflicting t-s-t Rule application involving the sink, and at most one s-t-s conflict involving the source. Finally, the source and sink are adjacent to at most two vertices, so the number of Index-1 Saddle Rule applications affected is clearly bounded.

For the s-t-s/t-s-t Rule, first consider the Consecutive Rule. We will give the argument for an application of the t-s-t Rule; the argument for an application of the s-t-s Rule is symmetric. There are two ways to conflict with a potential application of the Consecutive Rule: make one of the arcs involved "dirty" or remove one of the arcs involved. In applying the t-s-t Rule, no clean arcs are made dirty, so only the second case applies. The t-s-t Rule will remove at most two arcs out of a source, so at most four potential applications of the Consecutive Rule are affected (again, we don't count cases in which the arc operable by a subsequent rule is operable by the current rule). For t-s-t/s-t-s Rule conflicts with the Index-1 Saddle Rule, consider the case of an application of the t-s-t Rule. At most two arcs incident to the source are removed; the two sinks are combined. The only possible effects on potential Index-1 Saddle Rule applications are if the removed arcs are incident to index-1 saddles. No more than three conflicts are possible at each saddle. The argument for the s-t-s Rule is symmetrical.

For Consecutive Rule conflicts with potential Index-1 Saddle Rule applications, we note that the only way conflict can occur is through the removal of an arc incident to an index-1 saddle. Since a Consecutive Rule application removes exactly one arc that is incident to a source or sink, exactly one index-1 saddle can be affected. As in previous cases of conflict with the Index-1 Saddle Rule, there are at most three conflicts at that saddle.

Since the Index-1 Saddle Rule is applied last, there is nothing else to prove.

□

5.3 Proof of Main Lemma

We are now ready to show that the reduction algorithm runs in a logarithmic number of iterations of the main loop.

Lemma 5.3 [Main Lemma] *For any embedded connected planar DAG consistent with our invariants, the generalized reduction algorithm will work in $O(\log n)$ iterations of the main loop.*

Proof: This will follow if we show that the reduction algorithm removes a constant proportion of the arcs in each pass through the main loop.

Consider the graph at the start of the main loop. After some application-specific processing (which will not change the graph), the graph is cleaned up. Cleanup leaves the number of vertices, sources, and sinks unchanged. The number of arcs does not increase; therefore it is sufficient to show that we remove a constant proportion of the arcs left after cleanup. Lemma 4.3 implies that a constant proportion of these remaining arcs are operable. All that is left is to show that we remove at least a constant proportion of the operable arcs.

To show this, we will argue that the total number of arcs knocked out by conflicts is bounded by some constant times the number of arcs removed. In most cases this is obvious because the total number of interrule and intrarule conflicts is bounded by a constant. The exception is TB arcs (the first Type II conflict graph does not have bounded degree). However, we have argued above that a constant fraction of such operable arcs are removed, which implies that the total number of arcs knocked out by intrarule conflicts is at most a constant times the number of TB arcs removed. Since the number of interrule conflicts with TB arcs is bounded by a constant, the result holds. Since every operable arc is either removed or is subject to a conflict with an arc that is removed, this implies at least a constant proportion of the operable arcs are removed.

□

6 Applications

In this subsection we present an application that uses the abstract reduction procedure presented above. We also present the running time and number of processors needed to run this application.

6.1 Planar DAG Many-Source Reachability

The abstract reduction procedure can be used to solve the many-source reachability problem for planar DAGs. The problem can be stated as follows: given a planar DAG and an **initial set** of vertices in that DAG as the input, compute the set of vertices that are reachable via directed paths from the initial set. We will refer to the vertices reachable in this way as the **solution set**; we include the initial set as a subset of the solution set. Our solution to this problem consists of a set of application-specific actions taken at various points in the reduction algorithm; to show that it works we introduce invariants that allow us to prove that the result is correctly computed.

We introduce two flags at each vertex: a "reachable" flag indicating whether or not the vertex has been marked as reachable from one of the initial vertices, and an "active mark" flag that we will use to determine whether or not to propagate marks during the reduction phase. The algorithm starts with the input set of vertices having both their "active mark" and "reachable" flags set. We use the term **correctly marked** to indicate that a vertex in the solution set has its "reachable" flag set, and that a vertex not in the solution set does not.

The basic reduction algorithm combines vertices as the graph is processed. We need to keep track of such vertices while we compute reachability. Therefore we introduce the following terminology: A vertex in the current graph is an **original vertex** if it corresponds to exactly one of the vertices in the graph prior to the start of the reduction process (we will consider sources added during preprocessing to be original vertices). The remaining vertices in the current graph correspond to two or more vertices that have been combined by various reduction rules; we refer to them as **combined vertices**. For each combined vertex we refer to the original vertices that have been combined into it as its **components**.

For the purpose of proving that the algorithm for the reachability application works, we define the set of **active vertices**, which includes all original vertices that are not sources or sinks, plus any original sources that have active marks.

For the reachability application we keep track of the status of each vertex (combined or original).

We define a **reduction propagation step** as follows:

- If a vertex v is at the head of either a connectivity pointer or a directed arc such that the tail of that pointer or arc is a vertex with an active mark, v sets both of its flags (we say that the mark is propagated or passed over the arc or crosspointer). This rule also applies to internal vertices.
- If the directed arc over which a mark is passed is topological, all internal vertices of that arc are marked as reachable.
- If any internal vertex of a topological arc a receives a mark, the "active mark" and "reachable" flags of the head of a are both set.

- The “reachable” and “active mark” flags are unset for every sink and combined vertex.
- Any source that propagates an active mark unsets its “active mark” flag.

An **expansion propagation step** is defined similarly, except that all active vertices propagate their marks whether or not the “active mark” flag is set or not.

The application-specific processing added to the basic reduction algorithm is as follows:

- At the start of each cleanup phase d propagation steps are performed, where d is the degree limit introduced in Section 3.3. For each topological arc out of a source, if an active mark exists at an internal vertex higher than the high point, then the high point gets an active mark (this can be done in constant time in the CRCW model in constant time using the rank order on the topological arc). During the realignment phase, if a topological segment s of an arc out of a source is removed or replaced by a segment with no internal vertices, and if s contains a marked vertex, then the head of s is given an active mark (i.e., both flags are set).
- Whenever the TB Rule creates a topological arc a, if any internal vertex of a has an active mark, the head of a is given an active mark.
- Just prior to the application of specific rules, various numbers of propagation steps are done as follows:
  - One step is done before each of the Degree-1, Adjacent Degree-2 Source and Sink, s-t-s/t-s-t, and Consecutive Rules.
  - Two steps are done before the Unique-In/Unique-Out Contraction and Index-1 Saddle Rules.
  - For the TB Rule, one step is done prior to removing Type I TB arcs; two steps are done prior to removing each of Types II, III, and IV TB arcs.
- In rules where sources or sinks are combined with other vertices, the state of the vertices before combination is saved for the expansion phase, and the combined vertex is unmarked (i.e, neither of its flags are set).
- For the case of the Index-1 Saddle Rule in which the graph is split, the index-1 saddle vertex becomes a source. The active flag at this new source should be unset.

Between the reduction and expansion phases, each topological arc that was removed marks itself according to any marks at any of its vertices. More specifically, all vertices beyond the first vertex marked as reachable are marked as reachable.

The application-specific steps added to the algorithm for the expansion phase are as follows:

- One expansion propagation step is done after arcs are restored for the Unique-In/Unique-Out Contraction and the Index-1 Saddle Rules; two are done for the Degree-1 Rule.
- As TB arcs are added back to the graph their internal vertices may need to be marked. This involves checking crosspointers and checking the tail of the arc. If the tail has its “reachable”

flag set, all the internal vertices set their "reachable" flags. Otherwise, each internal vertex checks the lowest point that can reach it on each face it borders and sets its "reachable" flag accordingly. Also during expansion any vertices that became components of combined vertices are marked as necessary as they revert to original vertices. Note that the "active mark" flag is not used in this process. This step is done twice after Type IV TB arcs, twice after Type III TB arcs, twice after Type II TB arcs, and once after Type I TB arcs are restored.

At the end of the expansion phase we remove the restriction that sinks cannot be marked and do one more expansion propagation step to mark the sinks correctly.

It is easy to see that given the information on faces and topological arcs all of these application-specific actions can be done in constant time in the CRCW model.

The following lemma is useful in the proof that the reduction invariant holds through the cleanup phase (cleanup is discussed in Section 3.4, and some of the terminology used below is introduced there as well). A similar argument will be used to show that the expansion invariant holds through the reverse of the cleanup phase during expansion. For simplicity, in the text below we will refer to the highest points reachable from the frontier or beyond as "high points"; if no vertex is reachable from the frontier or below, then the frontier vertex is the high point.

Lemma 6.1 *For each topological arc a out of a source of degree $\leq d$, let v be the the highest internal vertex on a that both lies above the high point of a and is reachable from an active mark. Then during reduction, after $d-1$ mark propagation steps v is marked correctly.*

Proof: Such a vertex v is reachable only from marks that lie above the high points for this source, so we can prove this claim by looking at the subgraph consisting of the source and all arcs (or segments of arcs) out to the high point, and all pointers that lie between two vertices in this subgraph. Note that there is such a v for each arc in the subgraph that is reachable from an active mark. If v is the source, the result is trivial. If v already has an active mark, the result is again trivial. If the source does not have an active mark and v is not yet marked, then the last link in the path from any mark must be a crosspointer. In particular, there must be a crosspointer from v', the highest point reachable from a marked vertex on the other side of one of the adjacent faces: v must be at the head of a crosspointer from some vertex u reachable from a mark; if u is not the highest reachable vertex on its arc a', then the pointer rules indicate that the highest reachable vertex on a' must have a crosspointer to a vertex on a that is at least as high as v. Since such a crosspointer could not be to a higher point than v (that would contradict the fact that v is the highest point reachable from a mark), the crosspointer must be to v.

We continue extending this path of crosspointers back until it reaches a marked vertex. Note that the path can never backtrack to an arc that has previously been visited: no higher point on such an arc can lie on such a path (this contradicts the fact that the path includes only the highest reachable vertices); no lower vertex or one already on the path could lie on such a path because that would imply the existence of a cycle in the original graph, which is a DAG. Thus the path can have length at most $d-1$, and the phase of propagation across pointers will cause the highest points reachable from marks to be marked.

□

To prove that this marking process correctly marks the reachable vertices we use the following invariants, one for the reduction phase and one for the expansion phase. The reduction invariant is as follows:

Lemma 6.2 *During the reduction phase, the following two conditions hold:*

1. *There is no path from one active vertex to another through a vertex that is not in the active set (i.e., an original source with no active mark, a combined vertex, or a sink).*
2. *One or both of the following conditions hold for a vertex v in the active set if and only if v is in the solution set:*
   - *v is marked; or*
   - *there exists a path of arcs or crosspointers from an active mark at an active vertex to v, and the vertices on this path are all active vertices.*

Proof: The proof proceeds by induction. The base case is the initial graph. The first part of the invariant is obviously true since all vertices are in the active set. The second part of the invariant holds by the definition of the problem (note that the preprocessing adds only sources, so no added vertices violate the invariant).

For the induction step we consider the effects of the mark propagation steps, cleanup, and applying each rule in a single pass through the main loop. By the induction hypothesis, the invariant holds at the start of a pass through the main loop; by the argument below, it holds at the end as well, thus proving the lemma.

Cleanup: The first cleanup phase is application-specific processing, which for the current application consists of d rounds of mark propagation. Since mark propagation doesn't change any path in the graph or combine any vertices, the first part of the invariant obviously remains true.

It is also obvious that the second part of the invariant continues to hold because it holds prior to propagation by the induction hypothesis, and because marks are propagated only over paths of arcs and crosspointers through active vertices.

At this point any sources or sinks with degree higher than d (the cleanup degree constant) drop out of the cleanup process. Since they are unaltered by further cleanup steps, no changes to the invariant occur. We only need consider sources and sinks that are cleaned up.

The determination of the highest internal vertex on an arc out of a source reachable from the frontier (respectively lowest internal vertex on an arc into a sink that can reach the frontier) does not affect the invariant.

To show that the invariant holds after realignment, we first note that if v is an active vertex that lies between a high point and its cleaned source or below a low point and its cleaned sink, then v is removed. Second, it is straightforward to see that there is a path between two vertices after realignment only if there was a path between those vertices prior to realignment. In conjunction with the induction hypothesis and our previous arguments, this implies both that the first part of the invariant continues to hold, and that there is no path from an active mark to any active vertex not in the solution set (i.e., the second part of the invariant holds for vertices not in the solution set). Third, the second part of the invariant continues to hold for any marked active vertex. All that is left to show is that the second part of the invariant continues to to hold for unmarked vertices in the solution set.

Consider any path P from an active mark to a remaining unmarked active vertex such that P exists prior to realignment. Since the realignment actions don't disturb paths that don't include any vertices above high points or below low points, if P is such a path it will remain after realignment. By definition of low point, once a path reaches a vertex below the low point on an arc into some sink, all subsequent vertices on that path must be below the low point on some arc into that sink. Thus, P cannot pass through a vertex below the low point at any sink (recall that no vertices below low points remain after cleanup). The only remaining case to consider is if P passes through vertices above the high point at some source. By the definition of high point, such a path cannot include a vertex above a high point on an arc out of some source unless the path starts at such a vertex at that source. Thus if P is such a path it starts at an active mark above a high point at some source. Assume that this is the case. We need to show that any vertex on P that remains after realignment remains reachable from an active mark.

Note that Lemma 6.1 above implies that any high point h reachable by such an active mark at the same source will get an active mark as a result of the application-specific processing:

- either some vertex above h is reachable by such a mark, in which case the lemma shows that the highest reachable point above h will be marked, which implies h will be marked when marks are propagated to high points,
- or h will be the highest point on its arc a reachable by such a mark. In this case the last link on the path from the mark to h must be a crosspointer from some vertex on an arc a'. But then the crosspointer from u, the highest reachable point above the high point on a', must also have h as its head (it must point at least as high as h, but no higher point on a is reachable from such a mark). By the lemma, u has been marked after $d - 1$ propagation steps; then h will have been marked after the d propagation steps.

Thus the second part of the invariant holds for paths that go through high points.

If P does not go through a high point, there must be a first vertex v on the path that lies below a high point, and the path must follow a crosspointer from a vertex u above a high point to v. But this implies that there is a crosspointer p from u', the highest point marked on u's arc, to some point v' at or above v on v's side of the flow face. If v' is above the high point, the high point will be marked as per the previous paragraph. Otherwise Lemma 6.1 says that u' is marked by the time $d - 1$ propagation steps have occurred, so v' will have been marked by the time d propagation steps have occurred. Either way, the claim will hold.

TB Rules: The TB Rule does not combine vertices or create new paths, so the first part of the invariant continues to hold.

To show that the second part of the invariant continues to hold, we show that it holds after each step in the rule application/conflict resolution procedure.

The rule is first applied for Type I TB arcs. Note that the conflict resolution for this step ensures that at most one arc per face is removed in this step. We first consider the paths left after Type I TB arcs are removed. In particular, we want to show that for any pair of active vertices u and v that remain after Type I arcs are removed, if there was a path P from u to v prior to removal then there is a path P' from u to v after removal. Since the first part of the invariant holds at the time of removal, the path left after removal will include only active vertices. There are four cases to consider on the basis of how a removed Type I arc a is involved in the original path P:

- The path includes a. There are two possibilities: First, a may be replaced by a crosspointer, which replaces a in the path. Second, the tail of a may already have a crosspointer to a point above a's head on the other side of the face. In this case a will not be replaced by a crosspointer. However, there is a path from the tail of a to the head of a across the crosspointer and down the opposite side of the face. This path was in existence prior to the removal of a. Since the second half of the invariant held previously, and since the head and tail of a are active, all vertices on this path must be active. Since no other arcs on this face are removed, this path is not broken by the removal of any other Type I TB arc.

- The path enters the tail of a and leaves a via a crosspointer out of an internal vertex across face f. In this case we need to consider whether a is marked T or B on f (recall that Type I TB arcs are marked T on one adjacent face and B on the other, and that they are not both T and B on any face). If it is marked T, then the tail of a is the top of the face and there is a path from the tail of a to the head of the crosspointer along the opposite side of the face prior to a's removal. Since at most one arc per face is removed, this path is not affected by Type I arc removal. If it is marked B, the crosspointer at the tail of a points to a vertex on the opposite face as high or higher than the crosspointer involved in the original path. Thus using the crosspointer at the tail of a and part of the opposite side of the face boundary gives an alternative path; again, this path is not affected by Type I arc removal.

- The path enters an internal vertex of a via a cross pointer across face f and leaves via a's head. Let w be the internal vertex on a where the crosspointer enters. Again, we consider the cases in which a is marked T or B with respect to f. If it is marked T, then the crosspointer out of w on the other face f' adjacent to a reaches a point at or above the head of a (a is marked B with respect to f'); this provides that alternative path and is not affected by other Type I arc removals. If a is marked B with respect to f, then there is a path from the tail of the crosspointer to the head of a along the side of f opposite to a. As in previous cases, this path is not affected by removal of any other Type I TB arc.

- The path enters an internal vertex w of a via a crosspointer p across face f and leaves a via a crosspointer p' across face f' out of internal vertex w'. By the specification of the crosspointers, the crosspointer p'' out of w across f' reaches a point on the opposite side of f' that is as high or higher than the point reached by p'. Thus after arc removal the crosspointer that results from splicing p and p'' and possibly a segment of what was the opposite face of f' will provide the alternative path.

The only other problem that could occur during Type I arc removal is that an active mark at an internal vertex might be deleted when the associated arc is removed. We must show that this does not affect the invariant by leaving some unmarked active vertex in the solution set without a path from an active mark. We will show that this is prevented by the single step of mark propagation is done prior to Type I arc removal.

To see that any vertex reachable from an active mark is still reachable after the Type I TB arcs are removed, consider the situation just after removal. Any active mark removed must be at an internal vertex v of some topological arc a. There are two cases to consider. The first case is that the mark started at v. In this case the propagation rules insure that if the head is an active vertex, it will be marked with an active mark, which, given the argument above, implies that the invariant will continue to hold for any path from v through the head of the arc. The other paths out of v are via crosspointers. Note that the pointers out of v reach as high or higher than the crosspointers out of vertices lower than v on a. Thus for any path out of a lower vertex we can find a path out of a crosspointer at v and down the opposite side of some face; we only need to consider the crosspointers out of v. The heads of v's crosspointers are marked by the propagation phase, and since they lie on a face common to a, they are not removed when the Type I TB arcs are.

The second case to consider is when v receives an active mark as the result of the propagation phase. If the mark propagates in via the tail of a, then a's head is marked. Furthermore, since a source cannot be at the tail of a Type I TB arc, the active mark remains at the tail. Thus, any remaining vertex that was on the boundary of the face f for which a is marked T is reachable from this active mark. The only paths left to consider are those that leave v through a crosspointer on f', the face for which a is marked B. But the crosspointer out of a's tail reaches as high as the crosspointer out of v and provides a path from the active mark at a's tail to any vertex on the opposite side of f' reachable from v. Since no other arcs on f or f' are removed, the claim holds. If v received the active mark across a crosspointer p, then again the head of a is marked. If p is across face f, v's crosspointer on f is to a point below the tail of p since the graph is a DAG. Thus the only paths left to worry about are those that cross a second face f' of which a is on the boundary. All these paths would be via a crosspointer p' out of v. However, p and p' will be spliced into a new pointer that provides a path from the tail of p (where the active mark remains) to the head of p'.

The invariant therefore holds after Type I TB arc removal. We next consider the situation when Type II, Type III, and Type IV TB arcs are removed. These cases are similar and can be treated together.

We start with the useful observation that an arc of these types is marked both T and B on at least one face f, and that the conflict rules assure that no arcs are removed from the opposite side of f. Thus, when such an arc is removed a path from the tail to the head remains undisturbed.

We again have the situation that if a path from a vertex u to a vertex v exists before TB arcs of any of these types are removed, then it exists after the arcs are removed. The arguments to show this are similar to the to those used for Type I TB arcs. However, in this case there is the added complication that two such arcs connected by a crosspointer can be removed simultaneously. As a result, there are more cases to consider when an arc a is removed:

- The path enters the tail of a and exits the head.
- The path enters the tail of a and exits a crosspointer to a vertex that is not removed.
- The path enters a via a crosspointer from a vertex that is not removed and exits via the head.
- The path enters a via a crosspointer from a vertex that is not removed and exits via via a crosspointer to a vertex that is not removed.
- The path enters the tail of a and exits a crosspointer to a vertex on b, and exits via the head of b.
- The path enters the tail of a and exits a crosspointer to a vertex on b, and exits via a crosspointer to a vertex that is not removed.

- The path enters a via a crosspointer from a vertex that is not removed, exits via a crosspointer to a vertex on b, and exits the head of b.
- The path enters a via a crosspointer from a vertex that is not removed, exits via a crosspointer to a vertex on b, and exits b via a crosspointer to a vertex that is not removed.

The details of the arguments for these cases are similar to those for the Type I TB arc arguments, and are left to the reader.

The arguments that no active marks are lost are also similar to the arguments in the Type I TB arc case. There may be one additional crosspointer to deal with; however, the two propagation steps for these three types are sufficient to insure that the marks reach vertices that aren't removed. Once again the details are left to the reader.

This proves the claim for the TB Rule.

Degree-1 Rule: Once again we note that application of this rule doesn't create any new paths, so by the induction hypothesis and the preceding arguments, the first part of the invariant continues to hold. Also, no paths from active marks to vertices not in the solution set exist after rule application, so the second part of the invariant continues to hold for active vertices not in the solution set.

To show that the second part of the invariant continues to hold for active vertices in the solution set, we first consider a degree-1 source. Note that we can't assume that a_s, the arc out, is clean, because the application of the TB rules may have made the arc out a longer topological arc. Because all faces are flow faces, a_s is both the left path and the right path of the boundary of a flow face, and is the top arc on both sides of the face. Thus there are no pointers into a_s from any vertex outside it; such a pointer would be a backpointer and would imply that the initial DAG had a cycle, which is impossible. Thus the only paths between active vertices that involve internal vertices on a_s or its head are those that start at such a vertex. Therefore if there is no mark internal to the removed arc, the invariant continues to hold; if there is an active mark internal to the arc then any vertex reachable from the mark is reachable via a path through the head of the arc, which will get an active mark as a result of the propagation step for this rule application, or will have one already.

The case for a sink is simple. The arc a_t into a degree-1 sink is the bottom arc on both sides of a flow face. Therefore there can be no paths out of any vertices on a_t to higher points on the face because the graph started as a DAG. This implies there are no paths between any active vertices that will remain after this rule application through these vertices, and no such paths that start at these vertices. The arc a_t can be removed without affecting paths from active marks to remaining active vertices.

Unique-In(Unique-Out) Arc Contraction Rule: Note that a necessary condition for contraction of an arc to change the connectivity of the graph is that there be a path into some point below the tail of the arc and a path out of some point above the head of the arc. If the arc is not topological, this translates to a path into the head of the arc and a path out of the tail of the arc.

We first consider a unique-in arc a incident to a source s. Since this rule is only applied if a is clean, it is easy to show that the contraction of a doesn't change the connectivity of the graph: Because a is clean, there are no pointers into a or its head across the faces of which a is on the boundary, and, since a is the unique arc into its head, any pointers across any other faces into the head of a would be backpointers and contradict the fact that the graph is a DAG. By the condition stated above, contraction of a cannot create any new paths, so the first part of the invariant and the invariant condition on vertices not in the solution set continue to hold. It is also easy to see that if P is any path from an active mark to an active vertex such that neither s nor the head of a lie on P, then P is unaffected by the contraction. The only cases left are if either s or the head of a had an active mark prior to contraction. These cases are handled by the two propagation steps prior to rule application, which will either mark the vertices reachable from these vertices or will leave an active mark at an intermediate active vertex that is not affected by the application of this rule.

For sinks, a symmetric argument shows that no new paths are added, which implies that the first part of the invariant and the invariant condition on active vertices not in the solution set continue to hold. Also, any path that doesn't pass through the tail of the contracted arc is unaffected (the sink is never an active vertex and has no paths through it). Since the only paths through the tail of a must next cross a and terminate at the sink, the condition on active vertices in the solution set is unaffected.

Adjacent Degree-2 Sources and Sinks Rule: For this rule we a remove a source and a sink and their (clean) incident arcs. No new paths are created, so the first part of the invariant continues to hold, and the second part of the invariant continues to hold for active vertices not in the solution set. We also need to show no paths from active marks to active vertices are broken. The only such paths that can be broken are those that start at an active mark at the removed source, so the propagation step will insure that any vertex reachable from a mark at the source is either marked or reachable from an active mark at an intermediate vertex along the original path.

Source-Sink-Source (s-t-s)/Sink-Source-Sink (t-s-t) Rule: First consider the s-t-s rule. Two sources get combined into a single vertex, which will no longer be active. The only new paths created are those that start at one or the other of these sources, so no paths are created that violate the invariant condition for active vertices not in the solution set. Since there is a propagation step, neither of these sources will be active and the first part of the invariant will continue to hold. We also need to worry about breaking paths from active marks to active vertices in the solution set to complete the argument that the invariant holds for applications of this rule. But the only such paths that are affected by this rule are those that start at the sources (the sink is not in the active set, so removal of an arc into it doesn't break any such paths). The propagation step arguments used for previous rules apply here and give the desired result.

For the t-s-t Rule, two sinks get combined into a single vertex, which is not active. No new paths are created to any active vertex, so no paths are created that violate the invariant condition for active vertices not in the solution set, nor are any created that violate the first part of the invariant. We again only need to worry about breaking paths from active marks to active vertices in the solution set to complete the argument that the invariant holds for applications of this rule. But the only such paths that are affected by this rule are those that start at the source that loses an arc. The propagation step arguments used previously again apply and give the desired result.

Consecutive Rule: The arguments here are essentially the same as those for the s-t-s and t-s-t rules: a clean arc is deleted and two sources or sinks are combined. No new paths are created, so the first part of the invariant is unaffected, as is the condition on vertices not in the solution set. No paths from active vertices to other active vertices are affected with the exception of paths from active marks at sources involved in the rule application; as in previous cases, the propagation step for this rule will insure that the invariant still holds for active vertices in the solution set.

Index-1 Saddle Rule: There are two basic cases to consider: application with respect to sources and application with respect to sinks. In the situation where the rule is applied with respect to sources, there are two subcases: applications that involve contraction of arcs, and applications that separate the graph.

First consider the case of two distinct sources s_1 and s_2 with clean arcs into the saddle. By the same kind of arguments used in the Unique-In/Unique-Out Arc Contraction Rule, the only paths into the saddle are the two arcs out of the sources. If both arcs are contracted by the rule application, the same argument applies as for the Contraction Rule; if only one arc is contracted (w.l.o.g. assume the arc incident to s_1), then the only new paths created are those starting at s_2 and exiting the combined vertex via an arc that was out of s_1. However, since two propagation phases were performed prior to contracting the arc, s_2 no longer has an active mark (remember that the "active mark" flag at a source is unset after propagation) and thus is not an active vertex, so the invariant is not violated. The rest of the argument proceeds as for the Contraction Rule.

Next consider the case where the Index-1 Saddle Rule is applied at a source that is incident to the only two arcs into the saddle. In this case the two arcs from the source to the saddle are deleted and the graph is separated into two graphs. First consider the case for a degree-2 source. By previous arguments used for other rules in which arcs were only deleted, the first part of the invariant and the second part of the invariant's condition on active vertices not in the solution set will continue to hold since no new paths are created. If the source has degree 3, the tail of the third arc out of the source will become the former saddle, thus creating new paths. However, the former saddle becomes a source and (as part of the application specific processing) loses any active mark. Thus the new paths do not violate either the first part of the invariant or the second part of the invariant's condition on active vertices not in the solution set, which continue to hold. The rest of the argument is the same in the case of either degree-2 or degree-3 sources: For active vertices in the solution set we need to show that no paths from active marks to unmarked vertices are broken. Since the only paths broken by splitting the graph are those that go through the saddle, and since the only arcs into the saddle are from the source, we only need to worry about the case in which the source has an active mark. However, by the same arguments used above, the two propagation steps prior to rule application will insure that the invariant continues to apply for active vertices in the solution set, which are either marked or are reachable from some intermediate active vertex not affected by the rule application.

Third, consider the case of two distinct sinks t_1 and t_2 with clean arcs out of the saddle. By using arguments used for the Unique-In/Unique-Out Arc Contraction Rule, the only paths out of the saddle are into the sinks. If both arcs are contracted, no new paths are added, which implies that the first part of the invariant and the invariant condition on active vertices not in the solution set continues to hold. If only one arc is contracted, the only new paths created are those that extend a path into the combined vertex created from the contracted sink (say t_1) and the saddle. These paths all either end at the combined vertex or at t_2. Since sinks and combined vertices are not in the active set, the the first part of the invariant and the condition on active vertices not in the solution set are again unaffected. For active vertices in the solution set, no paths are broken and the second part of the invariant continues to hold.

Finally, consider the case where one sink is incident to the two clean arcs out of the saddle. In this case the two arcs from the saddle to the sink are deleted and the graph is separated into two

graphs. First consider the case for a degree-2 sink. By previous arguments used for other rules in which arcs were only deleted, the first part of the invariant and the invariant condition on active vertices not in the solution set will continue to hold since no new paths between active vertices are created. If the sink has degree 3, the head of the third arc into the sink will become the former saddle, thus creating new paths. However, the former saddle becomes a sink, which is not in the active set and will not be marked as part of the reduction application-specific processing. Thus the new paths do not violate either the first part of the invariant or the second part of the invariant's condition on active vertices not in the solution set, which continue to hold. The rest of the argument is the same in the case of either degree-2 or degree-3 sinks: For active vertices in the solution set we need to show that no paths from active marks to unmarked vertices are broken. Since the only paths broken by splitting the graph are those that go through the saddle, and since the only arcs out of the saddle are to the sink, it is clear that for active vertices in the solution set, no paths are broken and the invariant continues to hold.
□

At the end of the reduction phase, consider the active vertices. If we stop reduction when the graph is some constant size, all active vertices reachable from an active mark can be marked in constant time.

As noted above, between the reduction and expansion phases all removed topological arcs are correctly marked by propagation of any marks on internal vertices. Since the rank order of the vertices on the topological arc is known at the time of removal, we can use standard techniques to determine the first marked internal vertex in constant time in the CRCW model. All vertices can read this rank and mark themselves if they have a higher rank. Thus all this processing can be done in constant time in the CRCW model.

At this point the expansion phase begins. Expansion proceeds by reversing the reduction steps of the basic reduction algorithm, with application-specific steps added as specified above. Our moving of vertices between various sets for analysis purposes will also be reversed. Recall that during the expansion phase we will allow all marks at original vertices to propagate.

The expansion invariant is as follows:

Lemma 6.3 *During the expansion phase, all active vertices are correctly marked.*

Proof: This proof also works by induction on backward passes through the main loop. The base case follows from the discussion above and the following observations about the reduction procedure: First, since there are no active marks at vertices not in the active set, and since all mark propagation is from vertices with active marks to active vertices, the reduction invariant therefore implies that no active vertices are incorrectly marked.

We note that for the Source-Sink-Source (s-t-s)/Sink-Source-Sink (t-s-t) Rule, the Consecutive Rule, and the Adjacent Degree-2 Sources and Sinks Rule, the only change to the active set when these rules were applied in the reduction phase was that some sources dropped from the active set. In particular, sources that had an active mark propagated that mark out. By the argument above, these vertices were correctly marked prior to rule application, and thus are marked correctly when they are returned to the set of active vertices. Thus, in reversing these steps the expansion invariant remains unchanged.

For the Unique-In(Unique-Out) Arc Contraction Rule and the Index-1 Saddle Rule, we note that the only vertices that could be dropped from the active set when these rules are applied are those at the head (respectively tail) of an arc contracted into a source (respectively sink), plus any active source involved in the contraction. (In the case of the Index-1 Rule, these arcs may have been removed rather than contracted).

We start by considering the case where an arc a incident to a source was contracted. In expanding, if the source becomes active then by the definition of the active set it must have had an active mark prior to contraction, and is thus marked. Because the reduction invariant held at the time of contraction this mark is correct. If v, the head of a, becomes active, then since the reduction invariant held prior to contraction either v is marked correctly or there was an active mark at some vertex with a path (through active vertices) to v. But we proved above that v is reachable only from the one or two sources that have arcs into it, in which case if v was in the solution set but was unmarked prior to rule application, then there must have been an active mark at such a source. This mark would have marked v during the propagation step for this rule.

In the case of an arc a with tail v that was contracted with a sink t, we recall that sinks are never in the active set, and thus we only need to consider the case when v becomes active upon rule reversal. First consider the case when v is not in the solution set. Since the reduction invariant held prior to contraction, v is not marked, nor are any active vertices that have paths to v. Therefore after the expansion propagation step v is still unmarked. If v is in the solution set, either v was marked prior to contraction, or there was an active mark at some vertex u and a path form u to v through active vertices. This implies that prior to contraction there was an active vertex w with an arc (or perhaps a pointer) into v. By the induction hypothesis and the fact that w was not combined during this rule (i.e., it remained active), w is correctly marked when a is restored, so v is correctly marked during the subsequent expansion propagation step.

For cases in which the Index-1 Rule separates the graph, first consider the case of a source incident to both arcs into an index-1 saddle v. If after restoring these arcs v is active, we have the following cases:

- v is in the solution set and is already marked. The invariant obviously holds here.
- v is not in the solution set. We note that the only paths into v are the arcs from the source. Then the reduction invariant implies the sources can't be marked. Thus v is not marked by the propagation step and the invariant holds.
- v is in the solution set and not marked. By the reduction invariant that held prior to rule application, the source must have been marked and the propagation step prior to rule application would have marked v, so this case doesn't occur.

Now consider the case of a sink incident to both out arcs from an index-1 saddle v. Each copy of v (one in each of the graphs left after separation) became a sink and was subsequently inactive. If v becomes active when the rule is reversed, the situation is basically the same as in the case of expansion of a unique-out arc incident to a sink. In this case if v is in the solution set but unmarked, we can guarantee that an adjacent vertex remained active when this rule was applied (this follows from the conflict resolution procedure for the Index-1 Rule and the two steps of propagation done prior to applying this rule in the reduction process), and is now marked. This will insure that v is marked during the expansion propagation step.

Degree-1 Rule: When a degree-1 arc is restored, vertices internal to that arc may become active. Consider such an arc a. As was noted above in the argument for the reduction invariant, there are no crosspointers into the internal vertices on a. To see that the invariant continues to hold we consider the following sets of vertices that become active:

- Vertices in the solution set that are marked. It is obvious that the invariant holds for these vertices.
- Vertices in the solution set that are unmarked. Note that if these are internal vertices, at the time of their removal the only path to them was through a higher vertex on the arc. If a is incident to a source, then the reduction invariant implies that there must be a mark somewhere on a that was propagated to all reachable vertices in the step between expansion and reduction. If a is incident to a sink, then either the situation described above occurred, or the tail of a was active. In this case there is a marked vertex with an arc or pointer to the tail of a, and the two expansion propagation steps will insure that the vertices of this type are marked (the tail will become a sink when a is removed, so two steps are necessary).
- Vertices that are not in the solution set. Recall that the reduction invariant implies that these vertices are not marked, no higher vertex on the arc can be marked, and no vertex incident to an arc into any vertex on a can have a mark. Thus these vertices will remain unmarked.

TB Rules: As in previous cases, the reduction invariant allows us to argue that vertices not in the solution set will not be marked. Thus we need only consider restored active vertices that are in the solution set. Such vertices that are already marked are consistent with the invariant, so we only need to consider unmarked restored vertices in the solution set.

In the expansion phase the algorithm may restore a topological arc with internal vertices in the solution set that were not marked at the time of removal. To see that these vertices are properly marked after the propagation step following the arc's restoration, first note that any marks at internal vertices were propagated correctly between the reduction and expansion phases. Thus we only need to consider marks that come from outside the arc. By the reduction invariant, the only paths through which such marks can reach the arc must be through active vertices either at the tail of this arc or at the tails of pointers incident to internal vertices on the arc (as noted above, the marks could be at the tail of a path of pointers through internal vertices on restored arcs, but such paths can have length two at most). By the induction hypothesis these active vertices are correctly marked, so in the case where there is a mark at the tail of some restored arc, the internal vertices will be marked correctly by the expansion propagation step.

Now consider the case in which there is an unmarked vertex v on restored arc a such that v is in the solution set and the tail of a is unmarked. As noted above, there must be a path from some marked vertex that first crosses a crosspointer into a, then travels along some (possibly empty) segment of a to v. We now use the following fact, which is easy to prove: There is a marked vertex u on the opposite side of a face f from v such that there is a path from u to v of the form described above if and only if the lowest vertex across f that can reach v is marked. Therefore the marking process described in the application-specific processing will work (we note that the restoration of Type II TB arcs can break pointers into three pieces, so two steps are necessary in that case).

Cleanup: In reversing the cleanup process, the algorithm can restore some vertices that lie above high points or below low points to the active set. As in previous cases, vertices not in the solution set are not marked, nor will they be marked upon restoration. Thus we again only need worry about vertices in the solution set. If such vertices are marked, the reduction invariant implies they are correctly marked; thus we only need to worry about unmarked vertices in the solution set that become active.

For such vertices above high points, we argued above that the highest such points on each topological arc at a cleaned source were marked prior to removal. This implies that all lower vertices on the topological were marked in the step between reduction and expansion, so no unmarked vertices of the type we're considering remain above high points.

Thus we need only to show that unmarked vertices below low points and in the solution set are correctly marked after the d expansion propagation steps following the restoration of the previous graph structure. The argument is similar to that in Lemma 6.1. The claim follows by noting that for each arc containing a vertex that should be marked, there is a highest point at or below the low point that should be marked. If this highest point is already marked, we can propagate the mark along the topological arc to mark every active vertex on the arc as described below. If the highest point v reachable by a mark is not yet marked, then the last link in the path from a mark must be over a crosspointer. In particular, there must be a crosspointer from the highest point reachable from a marked vertex on the other side of one of the adjacent faces: there must be a crosspointer from a vertex u reachable from a mark; if u is not the highest reachable vertex on its arc, then the pointer rules indicate that the highest reachable vertex on u's arc must have a crosspointer to a vertex on v's arc that is at least as high as v. Since such a crosspointer could not be to a higher point than v (that would contradict the fact that v is the highest point reachable from a mark), the crosspointer must be to v.

We continue extending this path back until it reaches a marked vertex. Note that the path can never backtrack to an arc that has previously been visited: no higher point on such an arc can lie on such a path (this contradicts the fact that the path includes only the highest reachable vertices); no lower vertex or one already on the path could lie on such a path because that would imply the existence of a cycle in the original graph, which is a DAG. There are two cases to consider. First, the path works its way back to some marked vertex at or below the low point. Since no arc can appear in the path more than once, the path will have length at most d, and the phase of propagation across pointers will cause the highest points reachable from marks to be marked. The propagation of marks along topological arcs will mark the rest of the vertices on the arc. If the last vertex at or below a low point is not marked, then the path from a mark must go higher than the low point through some active vertex. In particular the path from a mark into this last vertex must be a crosspointer; the vertex at the tail of the crosspointer must be active (otherwise the reduction invariant would have been contradicted) and thus must be marked by the induction hypothesis. The low point on bottom arc of the side of the face below the marked vertex is not on this path by the construction since it will be marked or not active; thus the path is again of length at most d, and will be marked by the propagation steps.

To see that we can propagate marks at internal vertices along topological arcs, recall that we keep a rank ordering of the vertices on the topological arc. Thus, in the CRCW ARBITRARY model we can use standard techniques to determine in constant time the highest marked internal

vertex, and therefore in constant time we can have every lower vertex mark itself.
□

These invariants are sufficient to prove that at the completion of the algorithm the graph is correctly marked.

Theorem 6.4 *The marking procedure specified above solves the many-source reachability problem.*

Proof: By Lemma 6.3, every vertex in the graph is correctly marked at the end of the expansion phase except sinks. Thus, the last step of marking sinks will mark each sink if and only if it is in the solution set, and the theorem holds.
□

6.2 Running Time and Processor Count

The running time is determined by observing that the main loop is executed $O(\log n)$ times in the reduction and expansion phases. The running time of the main loop is dominated by the $O(\log^* n)$ time it can take to resolve conflicts for some of the reduction rules. Preprocessing time is dominated by the time for the main loop, so the running time is $O(\log n \log^* n)$ (this can be reduced to $O(\log n)$ through the use of randomization as noted above). The algorithm can be run using one processor per face, vertex, and arc, which is linear in the size of the input graph. When combined with Kao's strongly connected components algorithm [Kao93] the running time becomes $O(\log^3 n)$.

7 Reducing Planar Digraphs with Cycles

The techniques above can be expanded to work with planar graphs that have cycles. This is particularly useful in that we can then compute strongly connected components, and thus we can compute many-source reachability for any planar digraph (by first computing strongly connected components and then contracting them, then computing many-source reachability, then expanding back out the strongly connected components).

The reduction algorithm for the cyclic case is more complicated, as are the proofs of its correctness. We summarize some of the differences below:

- We must introduce two new rules (an arc contraction rule and an arc removal rule) for cycle faces. The structural invariant changes to allow cycle faces as well as flow faces.
- In addition to crosspointers on flow faces we must keep backpointers to the highest point reachable on the same side of the face.
- Cleanup is more complex because of the backpointers. We must now clean up two levels of arcs from sources or sinks. In addition, we must spend $O(\log n)$ time determining the connectivity implied by the backpointers during cleanup.
- The operability proofs must be modified to take into account the existence of cycle faces in the graph.

Details will be provided in a future Technical Report.

References

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, New York, 1976.

[Gaz91] H. Gazit. Optimal EREW parallel algorithms for connectivity, ear decomposition and st-numbering of planar graphs. In *Fifth International Parallel Processing Symposium*, May 1991. To appear.

[GPS87] Andrew Goldberg, Serge A. Plotkin, and Gregory Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 315–324, New York, May 1987. ACM.

[Kao93] Ming-Yang Kao. Linear-processor nc algorithms for planar directed graphs i: Strongly connected components. *SIAM Journal on Computing*, 22(3):431–459, June 1993.

[KK90] Ming-Yang Kao and Philip N. Klein. Towards overcoming the transitive-closure bottleneck: Efficient parallel algorithms for planar digraphs. In *Proceedings of the 22th Annual ACM Symposium on Theory of Computing*, Baltimore, May 1990. ACM.

[KS89] Ming-Yang Kao and Gregory E. Shannon. Local reorientation, global order, and planar topology. In *Proceedings of the 21th Annual ACM Symposium on Theory of Computing*, pages 286–296. ACM, May 1989.

[KS93] Ming-Yang Kao and Gregory E. Shannon. Linear-processor nc algorithms for planar directed graphs ii: Directed spanning trees. *SIAM Journal on Computing*, 22(3):460–481, June 1993.

[Lub86] M. Luby. A simple parallel algorithm fot the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, November 1986.

[MR85] Gary L. Miller and John H. Reif. Parallel tree contraction and its application. In *26th Symposium on Foundations of Computer Science*, pages 478–489, Portland, Oregon, October 1985. IEEE.

[MR89] Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. In Silvio Micali, editor, *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

[Phi89] Cynthia Phillips. Parallel graph contraction. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 148–157, Santa Fe, June 1989. ACM.

[RR89] Vijaya Ramachandran and John Reif. An optimal parallel algorithm for graph planarity. In *30th Annual Symposium on Foundations of Computer Science*, pages 282–287, NC, Oct-Nov 1989. IEEE.

[UY90] Jeffery Ullman and Mihalis Yannakakas. High-probability parallel transitive closure algorithms. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, Crete, July 1990. ACM.