Computer-Aided Design 44 (2012) 1235–1252

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Optimal uniformly monotone partitioning of polygons with holes

Xiangzhi Wei[a], Ajay Joneja[a,*], David M. Mount[b]

[a] Department of Industrial Engineering and Logistics Management, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

[b] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

ARTICLE INFO

Article history:
Received 15 November 2011
Accepted 11 June 2012

Keywords:
Polygon subdivision
Uniformly monotone subdivision
Simple polygons
Network flow
Planar graphs

ABSTRACT

Polygon partitioning is an important problem in computational geometry with a long history. In this paper we consider the problem of partitioning a polygon with holes into a minimum number of uniformly monotone components allowing arbitrary Steiner points. We call this the MUMC problem. We show that, given a polygon with n vertices and h holes and a scan direction, the MUMC problem relative to this direction can be solved in time $O(n \log n + h \log^3 h)$. Our algorithm produces a compressed representation of the subdivision of size $O(n)$, from which it is possible to extract either the entire decomposition or just the boundary of any desired component, in time proportional to the output size. When the scan direction is not given, the problem can be solved in time $O(K(n \log n + h \log^3 h))$, where K is the number of edges in the polygon's visibility graph. Our approach is quite different from existing algorithms for monotone decomposition. We show that in $O(n \log n)$ time the problem can be reduced to the problem of computing a maximum flow in a planar network of size $O(h)$ with multiple sources and multiple sinks. The problem is then solved by applying any standard network flow algorithm to the resulting network. We also present a practical heuristic for reducing the number of Steiner points.

1. Introduction

Subdividing a polygon into simpler polygonal components is a fundamental problem in computational geometry. Given a simple n-vertex polygon in the plane, the abstract problem is to subdivide the polygon's interior into a collection of components of a certain type. Perhaps the simplest example is polygon triangulation [1,2], which involves subdividing the polygon into triangles. Other examples include subdivisions into trapezoids [1,3] and convex polygons [4]. Even for components of a given type, there may be variations in the problem's formulation. For example, in some cases, the polygon may contain holes (that is, its boundary need not be simply connected), and sometimes additional vertices, called *Steiner points*, may be added as part of the decomposition. When Steiner points are allowed, there are two variants, *boundary Steiner points*, which may be added only along the polygon's boundary (meaning that the subdivision is formed by adding chords joining two points on the polygon's boundary) and *arbitrary Steiner points*, which may be placed anywhere. An excellent survey of the area can be found in [5].

Our focus will be on subdivisions that are called uniformly monotone. A polygonal chain is *monotone* with respect to a given direction, called the *scan direction*, if any line perpendicular to that direction intersects the chain in at most one point. A polygon is *monotone* with respect the scan direction if any line perpendicular to the scan direction intersects the polygon in at most one segment. A subdivision is *monotone* if each of its faces is a monotone polygon (possibly with respect to different scan directions). Such a subdivision is *uniformly monotone* if all of its components are monotone with respect to a common scan direction. (Formal definitions will be given in Section 2.) We consider the problem of decomposing a simple polygon into a minimum number of uniformly monotone components. We refer to this as the *MUMC* problem (minimum uniformly-monotone components). In our formulation the input polygon may contain holes, and we assume that (arbitrary) Steiner points are allowed. Uniformly monotone subdivisions are useful in a number of practical applications, including path planning [6] and in designing VLSI layouts [7].

The problem of monotone subdivision of polygonal objects has been well studied. If minimizing the number of components is not essential, it is known that in $O(n \log n)$ time it is possible to decompose a polygon with holes into $O(n)$ uniformly monotone components without the need for Steiner points [8,9]. For the problem of minimizing the number of components, the complexity depends on whether holes are present, Steiner points are allowed, and whether the subdivision is required to be uniformly monotone. For the case of Steiner-free, nonuniform monotone decompositions, Keil [10] presented an $O(Rn^4)$ time algorithm for the hole-free case, where R is the number of reflex vertices (that is, vertices whose interior angle exceeds π). He also showed that the problem is NP-hard if the polygon has holes.

* Corresponding author.
E-mail addresses: xzwei@ust.hk (X. Wei), joneja@ust.hk (A. Joneja), mount@cs.umd.edu (D.M. Mount).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.06.005

Table 1
Known results on subdividing polygon into monotone components.

Monotone subdivision	No Steiner points		With Steiner points	
	General	Uniform	General	Uniform
Polygon w/o Holes	$O(Rn^4)$ [10]	$O((n\log n)R^2 + nR^3 + R^5)$ [11]	Unknown	$O((n\log n)R^3 + R^5)$ [11]
Polygon w/ Holes	NP-hard [10]	NP-hard (see above)	Unknown	$O(n\log n + h\log^3 h)$ (ours)

Liu and Ntafos [11] considered the MUMC problem for simple polygons without holes. They showed that, given the scan direction, it is possible to solve the Steiner-free MUMC problem in $O(n\log n + nR + R^3)$ time. Their approach was to reduce MUMC to the problem of computing a maximum set of independent chords in a circle graph, where each node of the graph is a scan reflex vertex of the polygon and each chord is a visibility edge connecting a pair of such vertices (see Section 2 for definitions). However, this approach is not feasible for a polygon with holes, since the computation of a maximum set of independent chords is equivalent to computing a maximum independent set in an undirected graph, which is NP-hard [11,12].

They also presented an $O((n\log n)R + R^3)$ time algorithm for the variant where boundary Steiner points are allowed. These algorithms can be adapted to find the minimum decomposition over arbitrary directions by testing a finite number of judiciously chosen reference directions. Preparata and Supowit [13] showed that $O(R)$ reference directions suffice to determine the monotonicity of a polygon with respect to arbitrary directions. Based on this, Liu and Ntafos [11] showed that no more than $O(R^2)$ scan directions need to be considered for solving MUMC over all possible directions. This implies that the MUMC problem for polygons without holes can be solved in time $O((n\log n)R^2+nR^3+R^5)$ for the Steiner-free case and in time $O((n\log n)R^3 + R^5)$ for the case of boundary Steiner points.

In spite of these results, which are over twenty years old, it is remarkable that the computational complexity of MUMC has not been resolved for polygons with holes when Steiner points are allowed. In this paper we show that, given a polygon with n total vertices and h holes and a scan direction, the MUMC problem for arbitrary Steiner points is solvable in $O(n\log n + h\log^3 h)$ time. Because the number of Steiner points in an optimal solution may be as high as $\Omega(nR)$, this algorithm may actually run faster than the output size. This is possible because our algorithm produces a compressed representation of the subdivision of size $O(n)$. From this representation, it is possible to extract the decomposition in time proportional to its total size. The following table summarizes the current best known results, including our results presented in this paper (see Table 1).

As with Liu and Ntafos, our algorithm assumes that the scan direction is given. To solve the general MUMC problem, we apply a result of Arkin et al. [14], which shows that it suffices to test $O(K)$ reference directions, where K is the number of edges of the polygon's visibility graph. This implies that the general problem can be solved in time $O(K(n\log n + h\log^3 h))$. These results are presented formally in Theorem 2 of Section 5.4.

Our computational approach is quite different from that of Liu and Ntafos, which is based on dynamic programming. Instead, we show that in $O(n\log n)$ time the MUMC problem can be reduced to the problem of computing a maximum flow in a planar network with multiple sources and multiple sinks, such that the numbers of vertices and edges in this network are both $O(h)$. The result, which may be of independent interest, is presented in Theorem 1 of Section 5.4. The problem is then solved by applying a network flow algorithm to the resulting network. In particular, we apply the network flow algorithm of Borradaile et al. [15].

The rest of the paper is organized as follows. In the next section, we present definitions and preliminary observations.

In Section 3, we show that the notion of independent chords in [11] can be extended by generalizing chords into polygonal paths cutting through the polygon. This allows us to reduce MUMC to the problem of computing a maximum matching in an appropriate bipartite graph. In Section 4 we extend this result by presenting a reduction from MUMC to computing maximum flows in a sparse network of size $O(n)$. The network is essentially the dual graph of a modified trapezoidal map of the the polygon. In Section 5 we show how to reduce the size of the network to $O(h)$. Although our algorithm minimizes the number of components, we make no effort to minimize the number of Steiner points. In Section 6 we present a discussion of a practical approach for reducing the number of Steiner points.

2. Preliminaries

We begin by defining some standard terms. A *simple polygon* is the region bounded by a simple, closed polygonal chain. We allow the polygon to contain disjoint holes in its interior, each of which is a simple polygon. A *scan direction* is a unit vector in the plane, and a line perpendicular to the scan direction is called a *scan line*. Given a scan direction, a simple polygon is *monotone* with respect to this direction if any scan line intersects the polygon in at most one line segment. Although our algorithms can be applied to arbitrary scan directions (by the same methods used by Liu and Ntafos [11]), it will simplify the presentation to assume that, through an infinitesimal perturbation, the orthogonal projections of the vertices along the scan direction are distinct.

For the sake of illustration, we shall assume throughout the paper that the scan direction is parallel to the positive x-axis, and hence scan lines are parallel to the y-axis. A vertex of a polygon is *reflex* if its interior angle is greater than π. A *scan-reflex vertex* (SRV) is a reflex vertex both of whose incident edges lie on the same side of a scan line passing through the vertex. It is well known that a polygon is monotone with respect to the scan direction if and only if it has no SRVs [3,9]. Using the terminology of [3], if both incident edges of an SRV are to the left of this scan line, it is called a *merge SRV*, and if both are to the right of the scan line, it is called a *split SRV*. (These vertices are highlighted with small triangles in Fig. 1.) Throughout, R denotes the total number of reflex vertices, and r denotes the total number of SRVs.

Given a simple polygon, an *internal monotone chain* (or simply *monotone chain*) is an x-monotone polygonal chain that lies entirely within the interior of the polygon, with the possible exception of its endpoints. Two monotone chains are *independent* if they do not intersect, except possibly at their endpoints. It is not hard to show (see [3]) that an x-monotone subdivision can be induced by the addition of a collection of independent monotone chains, such that each merge SRV is the left endpoint of some chain, and each split SRV is the right endpoint of some chain. (Fig. 1(b) and (c) show two x-monotone subdivisions, with and without Steiner points, respectively.) A monotone chain that starts at a merge SRV or ends at a split SRV is said to *eliminate* this vertex. From the above discussion, we can view the monotone decomposition process as one of computing a collection of pairwise independent monotone chains in order to eliminate all the SRVs of the polygon. We call such a set an *independent eliminating set*. The following two lemmas show that it suffices to consider subdivisions induced in this manner.

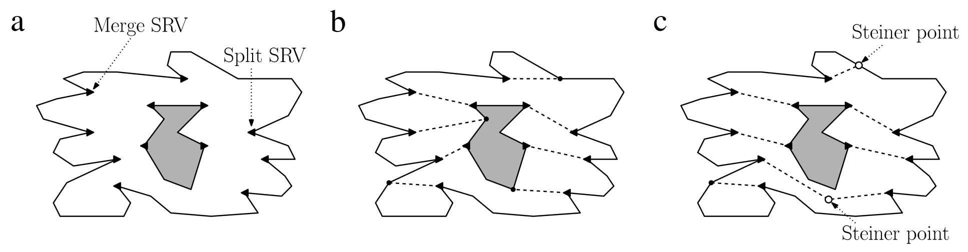


Fig. 1. Terminology: (a) a polygon with one hole, (b) a Steiner-free *x*-monotone subdivision with eight components, (c) an *x*-monotone subdivision (with both boundary and arbitrary Steiner points) with seven components.

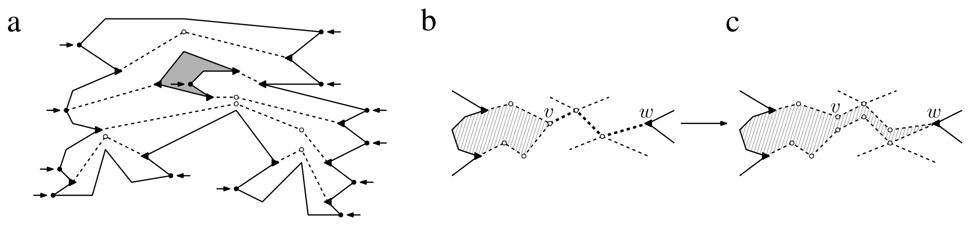


Fig. 2. Eliminating sets: (a) an *x*-monotone subdivision of a polygon with one hole with seven components, which is generated by an independent eliminating set of size seven, (b) a component (hatched) whose rightmost endpoint is not on *P*'s boundary, (c) extending the endpoint to *P*'s boundary.

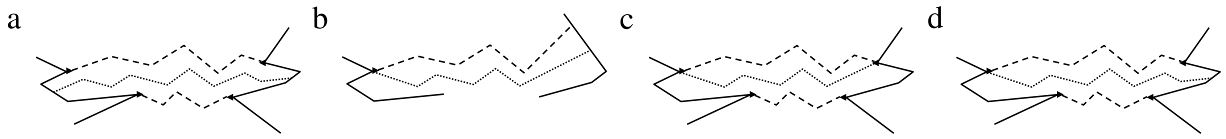


Fig. 3. Configurations of internal monotone chains that cannot occur in a minimal *x*-monotone subdivision: (a) neither of the endpoints of the dotted chain are SRVs, (b) two eliminating chains with a common endpoint and the other two endpoints are not SRVs, (c) two eliminating chains with a common endpoint and the other two endpoints are incident at the same SRV, (d) two eliminating chains with a common endpoint and one of the other two endpoints is an SRV while the other is not.

Lemma 1. *Let P be a polygon with h holes. Any x-monotone subdivision of P induced by an independent eliminating set of size $k + h - 1$ has exactly k components.*

Proof. We establish the relationship between the number of components and the size of the eliminating set. The boundary of P has $h + 1$ loops (one for P's outer boundary and one for each hole). Consider an independent eliminating set of some size c. Each time we add a chain from the eliminating set, we either create a connection between two disconnected boundary loops (and thus decrease the number of boundary loops by one), or we join two points on the same boundary loop (which induces a split in the monotone subdivision). Since exactly h chains result in the merging of boundary loops, the remaining $c - h$ chains cause monotone components to be split. Because we start with one component (the interior of P), the final number of monotone components is $c - h + 1$. Since $c = k + h - 1$, the number of components in the subdivision is equal to $(k+h-1)-h+1 = k$, as desired. □

Lemma 2. *Let P be a polygon with h holes. If P has a minimal x-monotone subdivision with k components, then there exists an x-monotone subdivision of P with k components that is formed by an independent eliminating set of size $k+h-1$. Furthermore, if any pair of eliminating chains of this set share a common endpoint, then their other two endpoints are distinct SRVs.*

Proof. Consider any minimal x-monotone subdivision. If this subdivision is formed from an independent eliminating set, then (by independence) the leftmost and rightmost vertices of each component both lie on the boundary of P. (These vertices are indicated by arrows in Fig. 2(a).)

Suppose that there is some component whose leftmost or rightmost vertex v does not lie on P's boundary (see the hatched polygon in Fig. 2(b)). For concreteness, let us assume that it is a rightmost vertex. (The other case is symmetrical.) Clearly, there exists an x-monotone path on the boundary of the subdivision emanating to the right of v that ends at a point w on P's boundary. As if ripping open a seam, we can split this path into two paths by duplicating each vertex on this path, excluding w, one slightly above the other (see Fig. 2(c)). Now the component's rightmost vertex is w, which lies on P's boundary. The number of components has not changed. By repeating this process, eventually the leftmost and rightmost vertices of all the components will lie on P's boundary.

We assert that the resulting subdivision is generated by an independent eliminating set. By walking from the leftmost vertex to the rightmost vertex along the top boundary of each component of the subdivision and removing the portions that are part of P's boundary, the result is a collection of x-monotone chains joining two boundary points. By our seam ripping, these chains are mutually independent. We need to show that each such chain either starts or ends at an SRV, i.e., it is an eliminating chain.

Suppose that this is not true and there exists a chain such that neither of its endpoints are SRVs, e.g., the dotted x-monotone chain in Fig. 3(a). Removing this chain will result in the union of the two x-monotone components sharing this chain as their common boundary. Clearly this union is an x-monotone component. But this reduces the total number of x-monotone components in the subdivision by one, contradicting our assumption that the original subdivision is minimal.

Next we show that if any pair of eliminating chains share a common endpoint, then their other two endpoints are distinct SRVs. Clearly, if the common endpoint is not an SRV, then the other two endpoints must be distinct SRVs, since each eliminating chain is incident to an SRV by definition. The other possibilities are shown in Fig. 3(b)–(d). In each of these cases, we can remove the

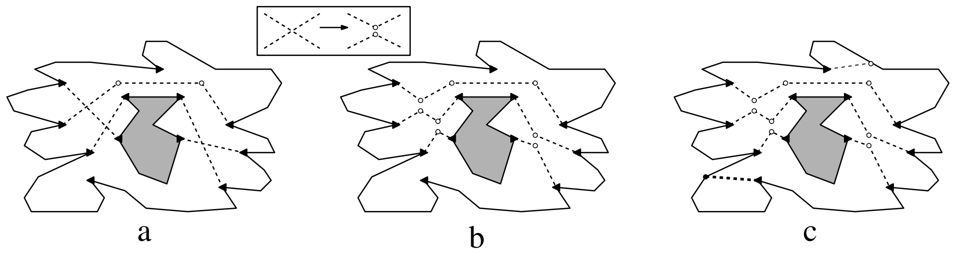


Fig. 4. Producing a monotone subdivision from a matching in F: (a) the initial set of five FEPS, (b) adding Steiner points and untwisting, (c) adding PEPs to complete the subdivision.

dotted eliminating chain and reduce the number of components by one, contradicting our assumption that the subdivision is minimal.

Let c be the size of the resulting independent eliminating set. By Lemma 1, the number of monotone components $k = c - h + 1$. It follows that $c = k + h - 1$, as desired. □

3. Minimum subdivisions and maximum matchings

In this section we demonstrate a connection between computing a minimum uniformly monotone subdivision of the polygon P and computing the maximum matching of a graph that encodes the monotone path structure within P. We have seen in Lemma 1 that it suffices to consider only subdivisions induced by independent eliminating sets. Each monotone chain appearing in such a set can eliminate at most two SRVs (one at each end). It follows directly that the number of chains needed in any optimal independent eliminating set is at least as large as the maximum of the number of merge SRVs and split SRVs, and it is not greater than their sum. Intuitively, it is clear that monotone chains that eliminate two SRVs are preferred over those that eliminate only one.

To make this intuition more formal, we define a *full eliminating path* (FEP) to be a monotone chain that starts at a merge vertex and ends at a split vertex, and we define a *partial eliminating path* (PEP) to be one that either starts at an SRV or ends at one, but not both. (The non-SRV end of a PEP may lie either on P's boundary or on another eliminating path.) Our approach is to define a bipartite graph $F = (V_M, V_S, E)$ that concisely encodes all the FEPs that *might* be used by the subdivision algorithm. The nodes of V_M correspond to the merge SRVs, the nodes of V_S correspond to the split SRVs, and the edges $E \subseteq V_M \times V_S$ consist of pairs (u, v) such that there exists an x-monotone chain from u to v in P.

Recall from graph theory that a *matching* in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that each vertex is incident to at most one edge of M [12]. Intuitively, in order to construct an independent eliminating set of minimum size, each chain of the set should eliminate one new merge SRV and one new split SRV. Such a set of FEPs would naturally correspond to a matching in F. We make this intuition more formal by showing that the MUMC problem (with arbitrary Steiner points) can be reduced to computing a maximum matching in F.

Lemma 3. *Consider a polygon P with h holes and r SRVs. If the size of a maximum matching in the bipartite graph F is m, then the size of a minimal x-monotone subdivision is $(r - h + 1) - m$.*

Proof. We first show how to construct a set of independent eliminating chains from a maximum matching in F. Let M be a maximum matching in F, and let $m = |M|$. Each edge of M corresponds to an FEP, but they need not be independent (see Fig. 4(a)). First, consider the subdivision induced by these FEPs, and let us assume (by a suitable perturbation) that whenever two such FEPs intersect, they do so transversally, that is, by crossing each other locally at a single point. For each such intersection point, we add two Steiner points and then perturb and *untwist* the paths so they do not intersect at this point (see Fig. 4(b)).

The resulting FEPs are pairwise independent. To complete the monotone subdivision, each unmatched SRV is eliminated through the addition of a PEP connecting it to any appropriate point on the boundary of P (see Fig. 4(c)). Clearly, this independent eliminating set satisfies the condition of Lemma 2. Therefore, this independent eliminating set is sufficient for the construction of a minimal x-monotone subdivision.

Next we show that the size of the x-monotone subdivision induced by this independent eliminating set is minimum. Recall that r denotes the total number of SRVs in P. The resulting independent eliminating set consists of m FEPs, which together eliminate $2m$ SRVs, and $r - 2m$ PEPs to eliminate the remaining SRVs. So it contains $m + (r - 2m) = r - m$ chains. Conversely, if P has an independent eliminating set of size $r - m$, then, in order to cover all r SRVs, m of these chains must be FEPs and the remaining $r - 2m$ are PEPs.

In summary, P has an independent eliminating set of size $r - m$ if and only if F has a matching of size m. By Lemma 1, P has an x-monotone subdivision with $k = r - m - h + 1$ components. Since $r - h + 1$ is constant for any given P, and therefore a minimal x-monotone subdivision of P has k components. □

The above lemma suggests a method for generating a minimal x-monotone subdivision. In practice, in generating each PEP, it is sufficient to use a line segment joining the corresponding SRV to any appropriate point either on the boundary of P or in the interior of any FEP (an example is shown later in Fig. 6(c)). We will not bother to discuss the implementation, since in the next section we will present a considerably more efficient approach.

4. Efficient construction by maximum flow

In this section and the next we present an alternate approach for solving the MUMC problem that is considerably more efficient than the approach described in the previous section. Rather than reducing the problem to computing a maximum matching in a potentially dense bipartite graph, we reduce the problem to computing a maximum flow in a sparse, planar graph. This approach has the nice feature that it provides a concise representation of the subdivision of size $O(n)$. The results of the previous section will be useful in establishing the optimality of this reduction.

We begin by adapting some terminology from the theory of network flows [12]. For our purposes, a *multi-supply, multi-demand network G* is a directed graph that has three types of nodes: *supplies*, *demands* (which together are called *terminals*), and *nonterminal nodes*. (Supplies and demands are more commonly called sources and sinks, respectively.) Each terminal node is incident to a single edge. For supplies, this edge exits the node, and for demands it enters the node. Each supply and demand node is associated with a finite nonnegative integer value called its *capacity*. Nonterminal nodes have effectively infinite capacity. Such a network is *planar* if

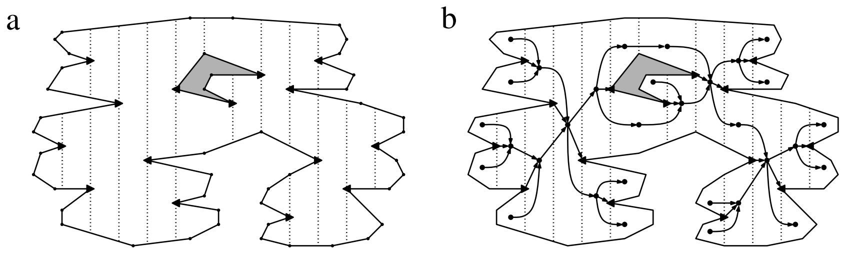


Fig. 5. Flow-based solution: (a) pseudo-trapezoidal map and (b) the associated flow network G.

the underlying directed graph is planar. A *flow* in such a network is a function f that maps each edge to a nonnegative real number, satisfying the following two requirements:

Capacity constraint: The flow on the edge incident to each terminal does not exceed its capacity.
Flow balance: For each nonterminal node, the sum of flows entering the node equals the sum of flows exiting the node.

The *total value* of the flow, denoted $|f|$, is the sum of the flows leaving the supply nodes (which, by flow balance, is equal to the sum of flows entering the demand nodes). The *max-flow problem* is that of computing a flow of maximum value in G. Because the capacities in our network are integers, it is well known that there exists a maximum flow such that the flow along each edge is an integer [12].

The first step of the reduction of MUMC to a flow problem involves computing a variant of a well known structure, called the *pseudo-trapezoidal map* of P. Given a simple polygon P, possibly with holes, the (standard) trapezoidal map is defined as follows. For each vertex of P, shoot a vertical bullet path from this vertex into the interior of P until it hits the boundary of P. Each SRV generates two bullet paths, one shot up and one down. For our purposes, it suffices to shoot bullet paths only from the reflex vertices (both SRV and non-SRV). These bullet paths subdivide P into a collection convex polygons, called *pseudo-trapezoids* (see Fig. 5(a)). The vertical segment separating two adjacent pseudo-trapezoids is called a *wall*. Because the number of walls is $O(R)$, it follows that the number of pseudo-trapezoids is $O(R)$. The pseudo-trapezoidal map can be constructed in $O(n \log n)$ time by a simple adaptation of the plane sweep algorithm given in [3].

Given the pseudo-trapezoidal map, we define a planar multi-supply, multi-demand network G as follows. There is one node of G for each pseudo-trapezoid and one node for each SRV (these are indicated by circles and triangles, respectively, in Fig. 5(b)). The merge SRV nodes are the supplies, the split SRV nodes are the demands, and the pseudo-trapezoid nodes are the nonterminals. The edges of G are directed and are of two types. First, for each pair of pseudo-trapezoids that share a common vertical wall, there is an edge directed from the one on the left to the one on the right. Second, each node associated with a merge (resp., split) SRV is joined to the node associated with the pseudo-trapezoid to its immediate right (resp., left). These SRV-trapezoid edges are also directed from left to right. We associate capacities with each of the nodes as follows. Each supply or demand node is assigned a capacity of one, and all the other (trapezoid) nodes are assigned a capacity of ∞. (In fact, a capacity of r is sufficient.) Observe that G is an acyclic, directed planar graph. The time to compute G is dominated by the time to compute the pseudo-trapezoidal map, which is $O(n \log n)$ [3].

Due to the unit capacities assigned to the nodes associated with each SRV vertex, each merge SRV can generate up to one unit of flow and each split SRV can receive up to one unit of flow. Intuitively, a collection of FEPs corresponds to a collection of paths in G, each starting at a merge SRV and ending at a split SRV. Such a collection of paths corresponds to an integer-valued flow in G, where the flow on each edge is the number of FEPs passing through the corresponding wall of the pseudo-trapezoidal map. This is made formal in the following lemma.

Lemma 4. *Given an n-vertex polygon P with holes, in $O(n \log n)$ time it is possible to compute a network G with $O(n)$ nodes and edges, such that there exists a matching in F of size m if and only if there exists flow in G of value m.*

Proof. The time to compute G and its size were discussed earlier. To relate the sizes of the maximum matching and maximum flow, observe that each FEP starts at some merge SRV of P, passes through some sequence of walls of the pseudo-trapezoidal map from left to right, and terminates at some split SRV. Therefore, each FEP corresponds to a path from a supply node to a demand node in G. A matching of size m in F corresponds to a collection of m such paths, with at most one path emanating from each merge SRV and at most one terminating at each split SRV. Thus, if we were to generate one unit of flow along each path of G corresponding to each monotone chain of the matching, we produce a flow of value m in G.

Conversely, let m be the maximum flow in G. As mentioned earlier, because the capacities are integers, the flow along each edge may be assumed to be an integer. By *flow decomposition* [16] any such flow of value m can be mapped to a set of m distinct paths, by peeling off one unit of flow along each supply-to-demand path that carries positive flow [12]. By our capacity constraints, there can be at most one path incident to any supply node and to any demand node. Therefore, the set of edges of F associated with these paths defines a matching in F, which completes the proof. □

By combining Lemmas 3 and 4, it follows that the MUMC problem can be reduced to computing a maximum flow in G. In the next section we present an efficient algorithm for doing this.

For the rest of this section we consider how, given a flow in G, to extract an independent eliminating set for P, where the number of FEPs in the eliminating set is equal to the flow value. Let m denote this flow value. The process involves two steps: first, computing the m FEPs of the eliminating set, which together eliminate $2m$ of the SRVs, and second, adding $r - 2m$ PEPs to eliminate the remaining SRVs. For the first step, suppose that we have already computed the maximum flow in G (see Fig. 6(a)). As mentioned above, the flow along each edge may be assumed to be a nonnegative integer.

Consider any pseudo-trapezoid τ of the map. The merge vertices (respectively, split vertices) separate the left (respectively, right) wall into one or more vertical segments. Each segment is associated with an edge e of G. Let f_e denote the flow on this edge. For each such segment, we create f_e Steiner points, called *portals*, spaced uniformly along this segment (see Fig. 6(b)). If this pseudo-trapezoid has a merge SRV incident to its left side, and the edge of G joining this merge SRV with τ carries a positive flow (which must be 1, because this is the node's capacity) a portal is created at this

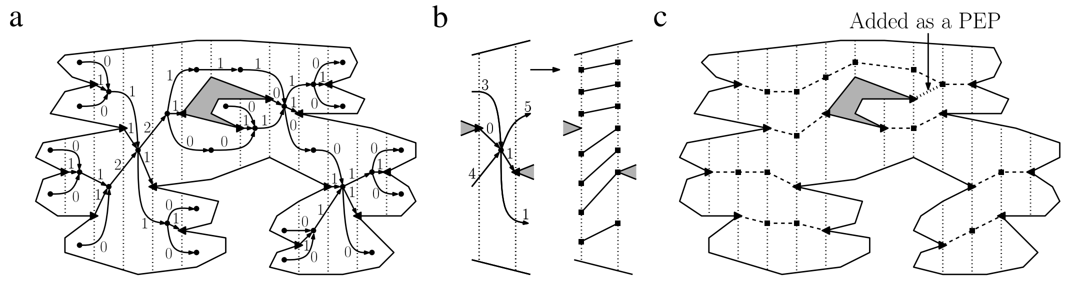


Fig. 6. Mapping a flow to a monotone subdivision: (a) a flow in *G*, (b) mapping of flows to portals and paths, (c) the final set of paths. (One PEP has been added at the end of the process.)

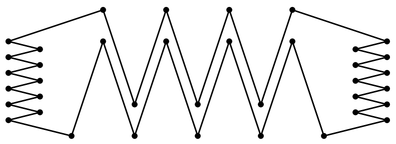


Fig. 7. An example showing that $\Omega(rR)$ Steiner points may be needed for a solution to MUMC (assuming a horizontal scan direction).

SRV. The same is done for any split SRV incident to the right side of τ that receives positive flow. It follows easily by flow balance that τ has an equal number of portals on each side. To form the final set of FEPs, an edge is generated from the *i*th portal on the left side of τ to the *i*th portal on the right. By convexity of the pseudo-trapezoids and the fact that we join corresponding portals on the left and right, all such edges lie within τ's interior and are pairwise disjoint (see Fig. 6(b)). By concatenating the resulting edges into paths, we obtain the desired set of *m* independent FEPs (see Fig. 6(c)).

To complete the construction, for each node v associated with a merge SRV that carries no flow (that is, one that has not been eliminated by one of the FEPs), let τ be the pseudo-trapezoid to its right. Observe that the right side of τ must contain either a portal or a vertex of *P* that is visible to this SRV (that is, the interior of the line segment joining v to this point does not intersect the boundary of *P* nor any FEP). A PEP is generated by adding a line segment joining v to any such visible portal or vertex of *P* (see Fig. 6(c)). A symmetrical process is performed for each split SRV that carries no flow. The resulting set of paths defines the desired solution to the MUMC problem.

Assuming that the flow is given, the time spent in the above construction is proportional to the sum over all the pseudo-trapezoids of the map of the flow passing through this trapezoid. As mentioned earlier, the map has $O(R)$ pseudo-trapezoids, and the maximum flow through each pseudo-trapezoid is $O(r)$. Therefore the total time for the construction is $O(rR)$. This also bounds the total number of Steiner points in the final monotone subdivision (Fig. 7 presents an example showing that this bound is tight. Later in Section 6, we will present a heuristic approach for further reducing the number of Steiner points).

The combinatorial complexity of the resulting subdivision could be as large as $O(n+rR) = O(n+R^2)$. We show next that it is possible to encode this subdivision much more concisely.

Lemma 5. *Given an n-vertex polygon P with holes and given a flow f in the associated network G, there exists a data structure of size O(n) from which it is possible to obtain any desired component of the associated x-monotone subdivision of P in time proportional to the size of the component.*

Proof. The data structure consists of the original polygon *P* and its pseudo-trapezoidal map, which are stored in a manner that permits local traversals of the pseudo-trapezoids of the map. Such data structures of size $O(n)$ are well known (for example, the double-connected edge list [3]). For each segment of the pseudo-trapezoidal map, we store the flow value of the associated edge of *G* (that is, the number of portals along the segment). We explicitly store all the PEP's. Since each PEP consists of a single edge that intersects a single pseudo-trapezoid, they can all be stored in space $O(r)$. The total size of this data structure is clearly $O(n)$.

Given this representation, the monotone chain of the independent eliminating set that emanates from any *single* SRV can be constructed as follows. Let us assume that this is a merge SRV, since the split case is symmetrical. First, if the SRV carries no flow, then the PEP associated with the SRV is output. Otherwise, each edge of the FEP is constructed by walking the chain through the map as follows. Since this is a merge SRV that carries flow, it contributes a portal to the left wall of some pseudo-trapezoid. The chain begins at this portal. Once the index of the portal on the left wall is known, in $O(1)$ time we determine the portal of this same index on the right wall, and then connect them by a line segment. We continue in this manner until arriving at the split SRV at the end of the chain. It follows that the time to output the entire chain is proportional to its size. Given this, it follows that we can traverse the boundary of any component in time proportional to its size. □

5. Efficiently computing the maximum flow

In the previous section we showed how to reduce the MUMC problem to computing a maximum flow in a directed planar network, *G*, where the network is derived from the pseudo-trapezoidal map of *P*. The network's size is proportional to *R*, the number of reflex vertices in *P*. In this section, we show that, through a collapsing process, it is possible to reduce this flow problem to one involving a planar network whose size is proportional the number of holes of *P*. Since the time needed to compute a maximum flow is superlinear in the size of the network, this can result in a significantly more efficient solution if the number of holes *h* of *P* is small relative to the number of reflex vertices *R*. (Note that the number of holes is never more than half the number of reflex vertices. This follows directly from the fact that each hole has at least two SRVs, corresponding to the leftmost and rightmost vertices of the hole.)

The process involves collapsing two types of substructures in the network and replacing each such substructure with an appropriate flow-equivalent structure of constant size. The first type of substructure, which is discussed in Section 5.1, is tree-like. We show that each of these structures can be modeled as a single node that is a net supplier or net demander of flow. The second type, discussed in Sections 5.2 and 5.3, is path-like. We shall see that each of these structures can be replaced by a chain of nodes of constant size. We will show that, after collapsing these two types of substructures, the resulting network has the same flow properties as the original network, subject to a fixed correction term. The total size of the collapsed network will be $O(h)$, and the maximum flow algorithm will then be applied to this collapsed network.

5.1. Collapsing subtrees

As mentioned above, the first part of the collapsing process is applied to substructures of the network that are tree-like in nature. We define these structures through an edge-marking procedure. In order to define this procedure, it will be convenient to ignore the directions on the edges of G, and treat G as an undirected graph. First, for each node of G of degree one, we mark its incident edge. Next, for each node u of G that has the property that all but one of its incident edges is marked, we mark u and this remaining edge. We refer to this last edge as u's *exit edge*. This procedure is repeated until no more edges are marked. (Fig. 8(a) shows the polygon P, the pseudo-trapezoidal map, and the (undirected) network G. Fig. 8(b) shows the network after the marking procedure with marked edges shown as solid lines and unmarked edges as broken lines.)

It is easy to see that, on termination of the marking procedure, there exists a (possibly empty) subset of the nodes X of G each of which is incident to two or more unmarked edges. Call these the *unmarked* nodes (shown as hollow points in Fig. 8(b)). It is also easy to see that any simple path in the network between two unmarked nodes consists entirely of unmarked edges (since any path formed by taking a marked edge from an unmarked node can lead only to marked nodes and marked edges). For each $u \in X$, define T_u to be the substructure of G consisting of all the nodes reachable from u by marked edges (shown in shaded regions of Fig. 8(b)). By the above observation, these substructures are disjoint from each other. Furthermore, each substructure is tree-like, where each exit edge connects a node to its parent. Clearly, u is the *root* of T_u. (If there are no holes in the polygon, then the dual graph is already a tree. In this case, we take u to be the last node to be marked, and T_u is the entire network.)

As we shall see, in addition to producing a certain amount of internal flow, each tree-like substructure can be modeled either as a net supplier or net demander of flow. The collapsing process will replace each substructure by either a single supply node or a single demand node, respectively. The capacity of this node is equal to the net flow supplied or consumed by the substructure. The computation of both the internal flow and net flow for any substructure is presented in Algorithm 1. The algorithm works recursively, propagating flow supply or demand to higher levels of the substructure along the exit edges.

The algorithm works by associating two quantities with each node u of the substructure. The first is the net flow supplied by u, denoted N_u, and the second is the internal flow generated within the substructure rooted at u, denoted I_u. The algorithm is implemented recursively. The initial call is made to the root of the tree, and recursive calls are made to the node's children. Information is passed up along the exit edges.

The pair (N_u, I_u) is computed for each node u as follows. First, suppose that u is a leaf node, that is, a node of degree one (see Fig. 9(a)). If u is an SRV, then $N_u = 1$, and otherwise, $N_u = 0$. In either case, $I_u = 0$. If u is an internal node of the tree, we make a recursive to call to each of its children. In order to determine the new internal flow generated at this node, we compute the sum of the net supply flows entering u from the left, denoted N^+, and the sum of the net demand flows exiting u to the right, denoted N^-. We route $\min(N^-, N^+)$ units of flow through u from the left to right. This quantity is added to the total internal flow. If u's exit edge is directed out of u, then u's subtree is a net supplier of the remaining $\max(0, N^+ - N^-)$ units of flow (see Fig. 9(b)), and otherwise is a net demander of $\max(0, N^- - N^+)$ units of flow (see Fig. 9(c)). If u is the root, there is no exit edge. In this case, we return $N^+ - N^-$ as the net flow. The sign indicates whether T_u is a net supply generator or net demand generator.

An example of the execution of Algorithm 1 is given in Fig. 10(a) and (b). Fig. 10(b) shows the values (N_i, I_i) for each node i after the

Algorithm 1: subtree-flow(u), computing the flow within a tree-like substructure T_u of G.

```
if (u is a leaf) then
    if (u is an SRV) then return (1, 0); // SRV leaf generates one unit of flow
    else return (0, 0); // non-SRV leaf generates no flow
else
    $N^- \leftarrow N^+ \leftarrow I \leftarrow 0$;
    for (each child v of u) do
        $(N_v, I_v) \leftarrow$ subtree-flow($v$); // recursively compute subtree flow
        if (v's exit edge enters u from left) then $N^+ \leftarrow N^+ + N_v$; // increment supply flow
        else $N^- \leftarrow N^- + N_v$;    // increment demand flow
        $I \leftarrow I + I_v$;    // increment total internal flow
    $I \leftarrow I + \min(N^-, N^+)$; // total internal flow in u's subtree
    if (u is the root) then return $(N^+ - N^-, I)$;    // return net flow
    else
        if (u's exit edge is directed out of u) then
            return $(\max(0, N^+ - N^-), I)$;    // return net supply flow
        else return $(\min(0, N^- - N^+), I)$;    // return net demand flow
```

algorithm has terminated. In particular, the values (N_u, I_u) for node u are computed from the labels on the children of u; notice that there is a net immediate demand of one unit, which can consume a supply of one unit out of the three units incoming (one unit from each child on its left). Therefore $N_u = 2$. The total internal flow consumed by its descendants to the left is two; adding this to the one unit consumed immediately at this node gives $I_u = 3$.

The collapsing process works by invoking Algorithm 1 on each of the root nodes u of the maximal substructures T_u. On return, I_u equals the total internal flow within T_u (shown as heavy solid lines in Fig. 10(c)). If $N_u = 0$, we collapse T_u to the single node u. If $N_u > 0$, we replace T_u by a single supply node directed into u with a capacity of N_u. (This case is shown in Fig. 10(c) and (d), where there are two unmatched flows from SRVs shown as hollow triangles.) If $N_u < 0$, we replace T_u by a demand node of capacity $-N_u$ and add an edge from u to this demand node.

An easy induction proof establishes that the flows generated by the algorithm are valid. This is formalized in the following lemma.

Lemma 6. *Given any node u that is the root of some substructure T_u, let (N_u, I_u) denote the result returned by Algorithm 1. Then there exists a flow of total value $I_u + N_u$ in T_u with I_u internal units of flow and N_u units of net supply (if $N_u > 0$) or $-N_u$ units of net demand (if $N_u < 0$).*

An important feature of the above algorithm is that it greedily accepts internal flow whenever it is discovered. The following lemma proves that this strategy is correct in the sense that any maximal flow in the original network can be modeled in the collapsed network, and vice versa. Let G be the original network, and let G' denote the collapsed network that results by applying the collapsing procedure to each tree-like substructure of G. Let I be the sum of the internal flows of all the substructures.

Lemma 7. *Given a network G and any node u that is the root of some substructure T_u, let (N_u, I_u) denote the result returned by Algorithm 1. Let G'_u denote the network that results by applying the subtree collapsing process to T_u. Given any flow f in G, there exists a flow in G'_u of value at least $|f| - I_u$. Conversely, given any flow f' in G'_u, there exists a flow in G of value at least $|f'| + I_u$. This latter flow can be computed in time proportional to the size of T_u.*

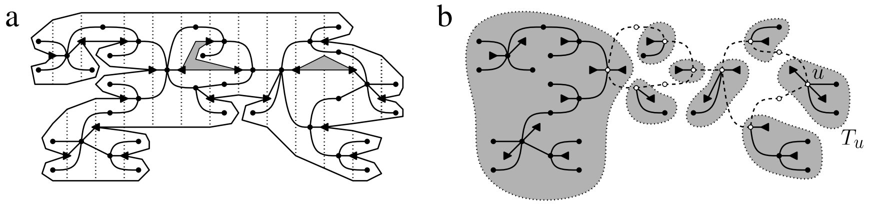


Fig. 8. Tree-like substructures of the (undirected) network G: (a) original network and (b) the substructures with the marked edges (solid), unmarked edges (broken), unmarked nodes (hollow), and subtrees T_u (shaded). (All edges are directed from left to right.)

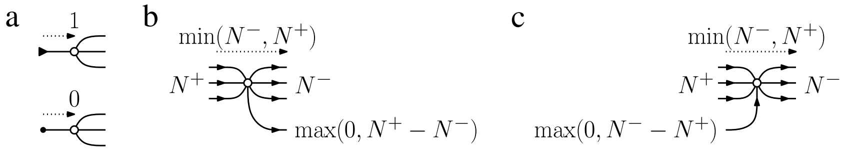


Fig. 9. The cases arising in the subtree-flow algorithm: (a) nodes of degree one, (b) internal nodes whose exit edge is directed out of u, (c) internal nodes whose exit edge is directed into u.

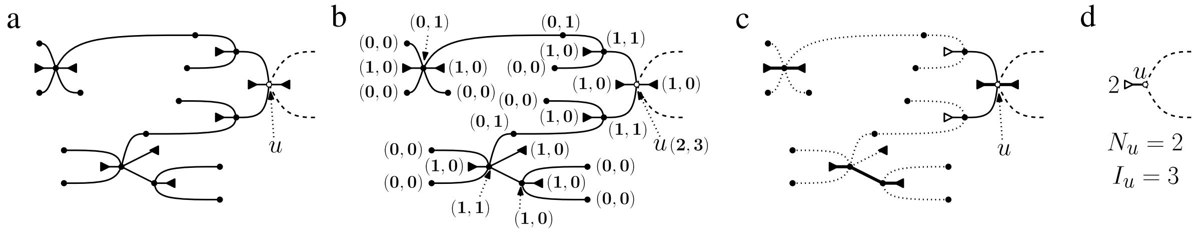


Fig. 10. Applying Algorithm 1 to a substructure with root u: (a) the substructure, (b) the (N, I) values, (c) the edges carrying positive net flow (solid lines), (d) the collapsed structure. (All edges are directed from left to right.)

Proof. We begin by establishing the lemma's second assertion. Consider any flow f' in G'_u. Let s_u be the supply or demand node associated with u as generated by the subtree collapsing process. Let us assume that s_u is a supply node (the demand case is symmetrical), let N_u be its capacity, and let f_N denote the flow assigned by f' to s_u's incident edge, which we think of as the net flow leaving the substructure. By flow capacity, $f_N \le N_u$. By Lemma 6, there exists a flow in T_u with I_u units of internal flow and N_u units of net flow through u. If $f_N < N_u$, we systematically reduce the flow provided by Lemma 6 by repeatedly finding a flow-carrying path from a merge SRV in T_u to u and reducing the flow along this path by one unit. This is repeated $N_u - f_N$ times, until the flow passing through u is equal to f_N. The resulting flow matches the net flow of f' through u, but has I_u additional units of internal flow. In order to implement this efficiently, rather than working one path at a time, we process each edge in $O(1)$ time, by simply reducing the flow on the edge according to the desired decrease in the number of paths. In this way, the total running time is proportional to the size of T_u.

Next, we establish the lemma's first assertion. Let f be any flow in G. Consider any unmarked node u of G, and let T_u be the associated substructure. We can decompose the flow f into three components, flows internal to T_u, flows external to T_u, and the net flow passing through u between T_u and the rest of the network. Let f_T^+ and f_T^- be the sums of flows entering and exiting u, respectively, from the edges of T_u. Let f_X^+ and f_X^- be the sums of flows entering and exiting u, respectively, from the other edges of G (see Fig. 11(a)). By flow balance, $f_T^+ + f_X^+ = f_T^- + f_X^-$, and hence, either $f_T^+ - f_T^- > 0$ (there is net flow leaving T_u) or $f_X^+ - f_X^- \ge 0$ (there is net flow entering u). Let us assume the former. (The other case is symmetrical.)

Define f_N to be $f_T^+ - f_T^-$, the net flow leaving T_u. Also, define f_I to be the total internal flow of f within T_u, which is the sum of the flows emanating from the merge SRV nodes of T_u minus the net flow f_N. (In Fig. 11(b), there are 5 units of flow emanating from the merge SRVs in T_u and $f_N = 4$ units of net flow leaving T_u, so $f_I = 1$.)

The flow generated by our algorithm is maximal in two respects. First, it maximizes the internal flow I_u. This can be seen by the fact that we generate internal flow greedily whenever possible. Hence, $I_u \ge f_I$. (Recall from earlier in Fig. 10(d) that $I_u = 3$.) Second, whatever flow cannot be resolved internally is passed up the tree whenever possible. Thus, it satisfies the property that, subject to the constraint of generating the maximum internal flow, it pushes the maximum residual flow through u. This implies that $I_u + N_u \ge f_I + f_N$, or equivalently, $(I_u - f_I) - (f_N - N_u) \ge 0$.

Given the flow f in G, we compute a new flow in G of greater or equal value as follows. First, we replace the flow assigned by f within T_u with the flow described in Lemma 6 (see Fig. 11(c)). This results in an increase of the internal flow by $I_u - f_I$, but may fail to satisfy flow balance at u. To remedy this, we consider two cases. In the first case, assume that $f_N < N_u$, that is, f pushes less flow through u than our algorithm. We systematically reduce the flow into u from N_u to f_N by finding a flow-carrying path from some merge SRV in T_u to u and reducing the flow along this path by one unit. This is repeated $N_u - f_N$ times. The resulting flow retains the $I_u - f_I$ additional units of internal flow, but now the flow through u is the same as in f. Because $I_u \ge f_I$, the modified flow value is $|f| + (I_u - f_I) \ge |f|$, as desired. The flow f' in G'_u is defined by pushing f_N units of flow through the supply node adjacent to u (which is legal, since its capacity is $N_u \ge f_N$).

In the second case, assume that $f_N > N_u$, that is, f pushes more flow through u than our algorithm. In this case we systematically reduce the flow leaving u from f_N to N_u, by finding a path starting at u that carries f's flow to some split SRV lying outside of T_u and reducing the flow along this path by one unit (see the upper right edge in Fig. 11(c) and (d)). This is repeated $f_N - N_u$ times. The flow through u, and hence the total flow value, has now decreased by

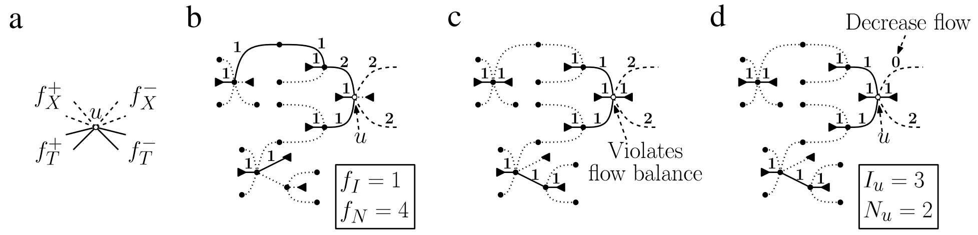


Fig. 11. Proof of Lemma 8: (a) classification of flows as internal (solid) or external (broken), (b) a sample flow, (c) a modified flow with internal flow I_u, (d) establishing flow balance. (All edges are directed from left to right, and edges that do not carry flow are shown as dotted lines.)

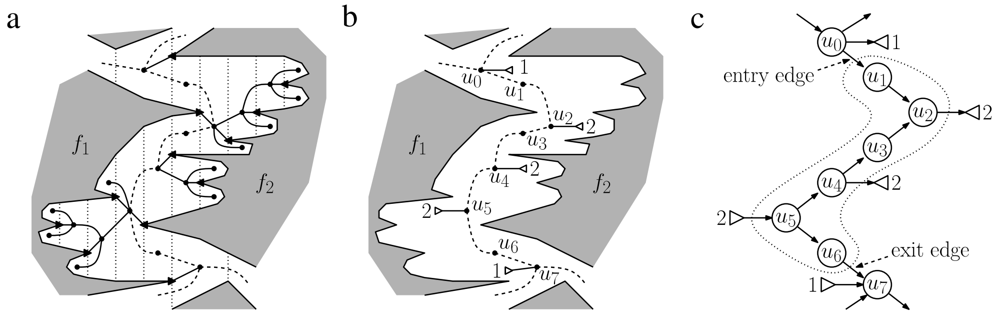


Fig. 12. Chains in the flow network: (a) a portion of the pseudo-trapezoidal map before subtree collapsing, (b) after subtree collapsing, (c) the resulting chain $\langle u_1, \ldots, u_6\rangle$. (Edge directions are from left to right.)

$f_N - N_u$. Combined with the increase of $(I_u - f_I)$ in internal flow, the new flow has value

$$|f| + (I_u - f_I) - (f_N - N_u) = |f| + (I_u + N_u) - (f_I + f_N) \geq |f|.$$

Observe that this is a legal flow, since we are pushing f_N units of flow through the supply node adjacent to u, which matches its capacity.

Thus, in either case, the total flow cannot decrease. By repeating this on each unmarked node u, the resulting flow has a total value at least as large as f. For each u, the flow passing between the supply or demand node s_u associated with u satisfies the node's capacity constraint. The flows generated within each substructure exactly match the internal flows generated by our algorithm. Thus, we may map this to a flow within G'_u whose total value is smaller by exactly the internal flow, I_u. Therefore, we have $|f'| + I_u \geq |f|$, which completes the proof. □

By applying the above lemma to each such substructure of G, we obtain the following.

Lemma 8. *Let G be a network, and let G' be the network that results by applying the subtree collapsing process to each substructure of G. Let $I_T(G)$ be the sum of the internal flows I_u over all nodes u that are the roots of some tree-like substructure. Given a flow f in G, there exists of flow in the collapsed network G' of value at least $|f| - I_T(G)$. Conversely, given any flow f' in G', there exists a flow in G of value at least $|f'| + I_T(G)$. This latter flow can be computed in $O(n)$ time.*

5.2. Collapsing monotone chains

Let us consider the network after collapsing all the tree-like substructures (see Fig. 12(a) and (b)). Each supply or demand node in the resulting network is generally the result of collapsing a subtree and hence has an associated nonnegative integer capacity. The next phase of the collapsing process involves collapsing path-like structures.

To define these structures, we begin by ignoring the directions on the edges of the network and ignoring edges incident to supply and demand nodes. Among the remaining nodes and edges, consider any sequence of nodes $\langle u_0, \ldots, u_{k+1}\rangle$ such that:

(1) all consecutive pairs u_{i-1} and u_i are adjacent,
(2) u_0 and u_{k+1} are of degree three or higher (ignoring supply and demand nodes), and
(3) the nodes u_1 through u_k have degree two (ignoring supply and demand nodes).

Given such a sequence, we define a *chain* to be the subsequence $\langle u_1, \ldots, u_k\rangle$ of degree-two nodes. (In Fig. 12(c), the original sequence is $\langle u_0, \ldots, u_7\rangle$, and the associated chain is $\langle u_1, \ldots, u_6\rangle$.) The edge (u_0, u_1) is called the chain's *entry edge* and (u_k, u_{k+1}) is called its *exit edge*. Intuitively, a chain corresponds to the region between two faces of P (for example f_1 and f_2 in Fig. 12(b)). In this section, we will show that each such chain can be replaced by a flow-equivalent chain of nodes of constant size.

Our approach for collapsing such chain structures involves two steps. In this section, we show how to reduce each chain to a special canonical form, called a *zig–zag chain*, in which the associated edges of the network alternate in direction from left to right. Next, in Section 5.3 we show how to replace each zig–zag chain with a flow-equivalent chain of constant length.

For the first step, consider an arbitrary chain $U = \langle u_1, \ldots, u_k\rangle$ of the network. The edge connecting two consecutive nodes u_{i-1} and u_i is a *forward edge* if it is directed as (u_{i-1}, u_i), and otherwise it is a *backward edge*. We can decompose any chain into *maximal monotone subchains*, where each subchain consists entirely of forward edges (see Fig. 13(a)) or entirely of backward edges (see Fig. 13(b)). (In our earlier example, the chain in Fig. 12(c) consists of two forward monotone subchains, $\langle u_1, u_2\rangle$ and $\langle u_5, u_6\rangle$ and one backward subchain $\langle u_2, \ldots, u_5\rangle$.)

It will simplify the presentation for now to assume that each node u_i of a monotone subchain is associated both with an adjacent supply node of capacity s_i and a demand node of capacity d_i. If no such node exists, we set s_i or d_i to zero. We will sometimes abuse notation by talking about s_i as a supply node and d_i as a demand node. Note that the directions of the entry edge and exit edge cannot generally be inferred. (In Fig. 12(c) for example, the chains $\langle u_1, u_2\rangle$ and $\langle u_5, u_6\rangle$ are both maximal monotone forward chains, but one has a forward entry edge and the other a backward entry edge.)

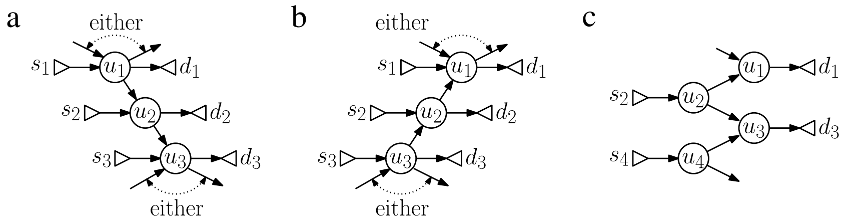


Fig. 13. Chain examples: (a) a maximal forward monotone subchain, (b) a maximal backward monotone subchain, (c) a zig–zag chain.

We begin by showing how to collapse each maximal monotone subchain $U = \langle u_1, \ldots, u_k \rangle$ into a single-edge chain. (After this, all chains will be zig–zag chains, which we will deal with later.) By symmetry, it suffices to consider just the case of a forward chain. Because the supply and demand nodes adjacent to u_1 (resp., u_k) can route flow through the entry edge (resp., exit edge), and we do not know the optimal routing, we will ignore them from our processing for now. On the other hand, the supply and demand nodes adjacent to u_2 through u_{k-1} can only route flow in one direction. Our approach will be to satisfy each of the demand nodes d_j, for $2 \le j \le k-1$, by drawing supply from the nearest preceding supply node, that is, from s_i where $2 \le i \le j$. Intuitively, this choice is justified since we know that the flow must come from higher up along the chain, and there is no harm in taking it from the closest such supply node.

For the sake of efficiency, our algorithm maintains the supply nodes having positive net supply on a stack (see Algorithm 2). The stack is ordered so the closest unexhausted supply node to the current demand node is on top and so is accessible in constant time. Whenever a new flow is generated, we decrement the associated supply and demand values, and increase a counter I of the total internal flow generated in the process. When a node's supply is exhausted, it is popped from the stack.

Algorithm 2: Computing the flow within a forward monotone subchain $U = \langle u_1, \ldots, u_k \rangle$.

```
for $j \leftarrow 2$ to $k-1$ do
    push $j$ onto stack;    // push supply node $s_j$ onto the stack
    while (stack is nonempty and $d_j > 0$) do // try to satisfy $d_j$'s demand
        $i \leftarrow$ top of stack;
        if ($d_j \ge s_i$) then    // demand exceeds remaining supply
            $d_j \leftarrow d_j - s_i$; $I \leftarrow I + s_i$; $s_i \leftarrow 0$; pop stack; // $s_i$ is now exhausted
        else
            $s_i \leftarrow s_i - d_j$; $I \leftarrow I + d_j$; $d_j \leftarrow 0$;    // $d_j$ is now satisfied
```

The process is illustrated in Fig. 14. The initial monotone subchain is shown in Fig. 14(a). The procedure starts by attempting to satisfy the demand of d_2, which it does by removing 2 units of flow from s_2 (reducing its value to 4). Next, d_3 tries to satisfy its demand. At this time, the stack contains $\langle s_3, s_2 \rangle$ with s_3 on the top. It first takes 1 unit from s_3 (thus exhausting it and popping the stack), next it takes the remaining 4 units from s_2 (also exhausting it and popping the stack). At this point $2 + 1 + 4 = 7$ units of internal flow have been generated (see Fig. 14(b)). The process continues until we have processed all the demand nodes from d_2 to d_5. The procedure terminates with $I = 22$ total units of internal flow generated (see Fig. 14(c)).

Let us consider the properties of supplies and demands after termination. Define the *core* of the subchain to consist of the nodes u_i, for $2 \le i \le k-1$ and the associated supply and demand nodes. Let s'_j and d'_j denote the core supply and demand values that result on termination of the algorithm. It is easy to see that Algorithm 2 implicitly defines a flow through the core, whose internal flow value is I, and whose residual supply and demand values are given by s'_i and d'_j.

Let $S' = \sum_{i=2}^{k-1} s'_i$ and $D' = \sum_{j=2}^{k-1} d'_j$ denote the core's total residual supply and residual demand, respectively. Let ℓ be the largest index in the interval $[2, k-1]$ such that $d'_\ell > 0$, or $\ell = k$ if no such index exists. (In Fig. 15(a), $\ell = 3$.) We refer to the portion of the core lying on or above level ℓ to be the *upper core*, and the rest is the *lower core* (see Fig. 15(a)). By definition of ℓ, all the demands of the lower core are zero. It is also easy to see that all the supplies in the upper core are zero. The reason is that each such supply s'_i can reach d'_ℓ, and since d'_ℓ is not exhausted, s'_i must be. Thus, all the nonzero residual supplies in the core lie in the lower core and all the nonzero residual demands lie in the upper core.

We can say more about the flows in the upper and lower cores. Returning to the original supplies and demands, define the total upper supply to be $S_U = \sum_{i=2}^{\ell} s_i$, and define D_U, S_L, and D_L analogously for the total supplies/demands from the upper/lower cores. (For example, in Fig. 15(b), $S_U = 6+1 = 7$, $D_U = 2+9 = 11$, $S_L = 17+5 = 22$, $D_L = 3+12 = 15$.) Observe that any internal flow generated by our algorithm that emanates from a supply node in the upper core may only satisfy demands in the upper core. This is because, in our algorithm, demand is satisfied from the closest available supply, and d_ℓ has not been fully exhausted. It follows that, D' is equal to the excess demand present in the upper core, that is, $D' = D_U - S_U$. (For example, in Fig. 15(a), we have $D' = 4$.) Symmetrically, all the demands of the lower core are satisfied by supplies in the lower core, and so, $S' = S_L - D_L$. Thus, we can partition the internal flow I generated by our algorithm into two parts, those in the upper core and those in the lower core. The total internal flow for the upper core is S_U, and the total internal flow for the lower core is D_L. (For example, in Fig. 14(a), we have $I = S_U + D_L = 22$.)

We assert that, given any flow f in the original network, there exists a flow f' of greater or equal value that satisfies the basic structural properties as the flow produced by Algorithm 2.

Lemma 9. *Let G be the network resulting after subtree collapsing, and let $U = \langle u_1, \ldots, u_k \rangle$ be a monotone subchain in G. Let I, S', and D' denote the internal flow, residual supply, and residual demand as generated by Algorithm 2. Given any flow f in G, there exists a flow f' of greater or equal value in G, that satisfies the following properties, where f_i and f'_i denote the flows of f and f', respectively, along the edge (u_{i-1}, u_i):*

(i) *the internal flow of f' within the core is I,*
(ii) $f'_{k-1} - f'_1 \le S'$, *and*
(iii) $f'_1 - f'_{k-1} \le D'$.

Proof. We may view the internal flows generated by Algorithm 2 as a multi-set of paths, each carrying one unit of flow from some supply node of the core to some demand node of the core. By the

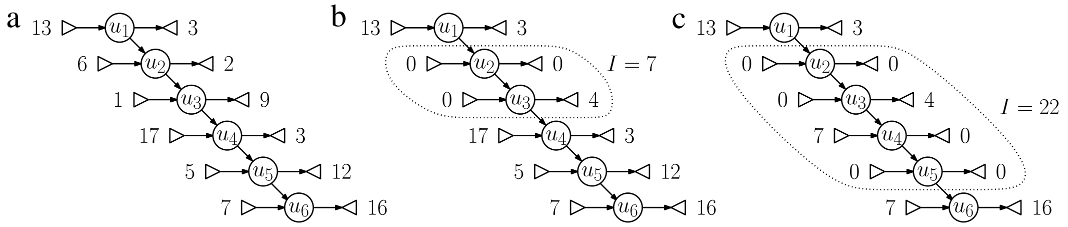


Fig. 14. Flow in a monotone chain: (a) a maximal forward subchain, (b) after two iterations of the for-loop of Algorithm 2, (c) final result.

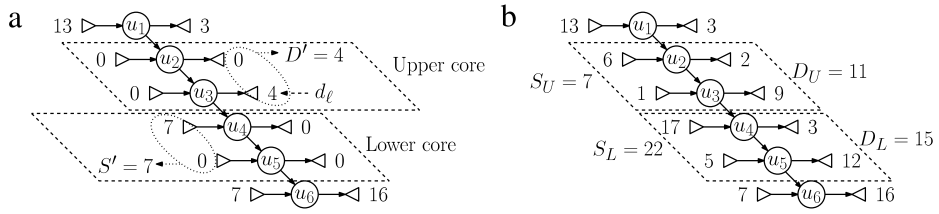


Fig. 15. The upper and lower cores: (a) the residual supplies S' and demands D', (b) the upper and lower supplies and demands in the original subchain.

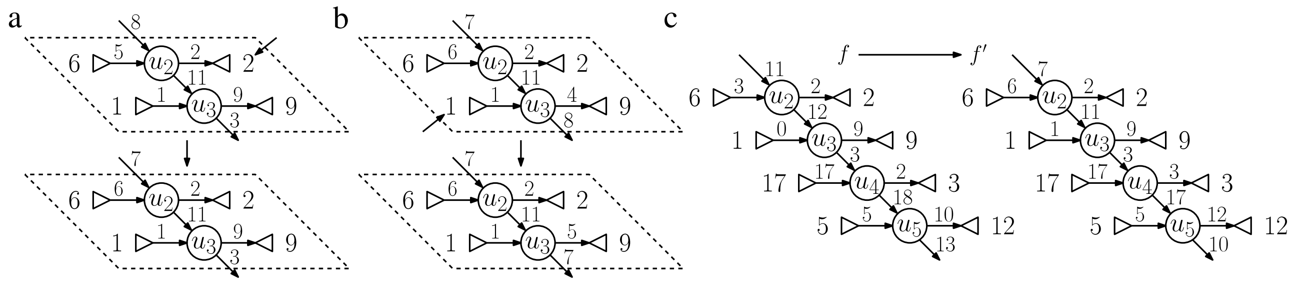


Fig. 16. Proof of Lemma 9, where $\ell = 3$, $D' = 4$, and $S' = 7$: (a) increasing internal flow in the upper core so that $f_1 - f_\ell \le D'$, (b) increasing internal flow in the upper core so that $f_\ell - f_1 \le 0$, (c) an example of the entire process.

remarks made earlier, this multi-set can be partitioned into two subsets, paths lying in the upper core and those in the lower core.

We will establish the assertion in two parts, by considering the upper and lower cores separately. First, suppose that $f_1 - f_\ell > D'$. By viewing flows as paths, this means that there are more than D' paths in f's flow that pass through the entry edge but not through the upper core's exit edge, $(u_\ell, u_{\ell+1})$. These paths must terminate at the demand nodes of the upper core. Algorithm 2 generates S_U units of internal flow in the upper core, each of which terminates at some demand node in the upper core. As observed earlier, $D' = D_U - S_U$. Therefore, $(f_1 - f_\ell) + S_U > D' + (D_U - D') = D_U$. That is, the sum of the flows from f and the internal flows from Algorithm 2 exceed the demand capacity of the upper core. Thus, there must be at least one upper core demand node that receives flow from both f and the internal flow of Algorithm 2. (In Fig. 16(a), d_2 is such a node.) We remove from f the path to this node, and add the internal flow to this node generated by the algorithm. This decreases f_1 but does not alter f_ℓ (see Fig. 16(a)). We repeat this process until $f_1 - f_\ell \le D'$.

Next, considering just the upper core again, suppose that $f_\ell - f_1 > 0$. The associated flow paths of f originate at some supply node of the upper core and pass through the upper core's exit edge. Algorithm 2 generates S_U units of internal flow in the upper core, each of which originates at some supply node in the upper core. We have $(f_\ell - f_1) + S_U > S_U$, implying that there must be at least one upper core supply node that initiates flow for both f and the internal flow of Algorithm 2. (In Fig. 16(b), s_3 is such a node.) We remove from f the path originating at this node (removing the portion of the path lying within the lower core as well), and add the internal flow originating from this node generated by the algorithm. This decreases f_ℓ but does not alter f_1 (see Fig. 16(b)). We repeat this process until $f_\ell - f_1 \le 0$.

By applying a symmetrical transformation on the lower core, we may modify f so that $f_{k-1} - f_\ell \le S'$ and $f_\ell - f_{k-1} \le 0$. (This may result in a violation of the previously established conditions for the upper core, which we fix by applying the upper-core modifications again. Because each operation strictly decreases the flow value, this process must eventually terminate.)

Ignoring its internal flow, the resulting flow has the following properties. First, its net increase is

$$f_{k-1} - f_1 \le (f_{k-1} - f_\ell) + (f_\ell - f_1) \le S' + 0 = S'.$$

Similarly, its net decrease is

$$f_1 - f_{k-1} \le (f_1 - f_\ell) + (f_\ell - f_{k-1}) \le D' + 0 = D'.$$

It also carries f_ℓ units of flow directly through from f_1 to f_{k-1}.

In contrast, the flow generated by Algorithm 2 consists only of internal flow, and has up to D' units of residual demand in the upper core and up to S' units of residual supply in the the lower core. Therefore, we may replace f within the core with an equivalent or greater flow as follows. First, use the I units of internal flow generated by our algorithm. Next, generate f_ℓ units of flow, which pass directly through from f_1 to f_{k-1} (which we can do because these nodes have infinite capacity). Then, generate an equivalent amount of flow in the upper core to match f's net decrease (which we can do from the D' units of residual demand). Finally, generate a equivalent amount of flow in the lower core to match f's net increase (which we can do from the S units of residual supply).

Because the internal flow saturates every supply node in the upper core and every demand node in the lower core, the internal flow is as large as possible, and so the value of this flow is at least as high as f's. Clearly, it is feasible, and it satisfies properties (i)

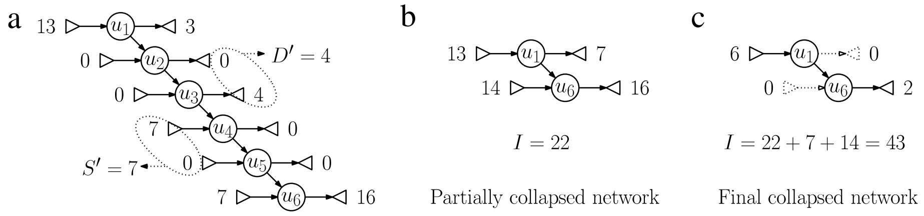


Fig. 17. The subchain collapsing process.

through (iii) above (see Fig. 16(c)). Finally, define the final flow f' by taking the above flow within the core and using the original flow f throughout the rest of the network. □

The subchain collapsing process continues as follows. Observe that the core's residual demand D' can only be satisfied by the flow coming through u_1. Similarly, any flow generated by the the core's residual supply must pass through u_k. Therefore, we may collapse the subchain by removing the core entirely, and replacing d_1 with a demand node of capacity $d_1'' = d_1 + D'$, and replacing s_k with a supply node of capacity $s_k'' = s_k + S'$. Call the resulting network the *partially collapsed network* (see Fig. 17(b)).

Observe that this may generally result in having u_1 being adjacent to both supply and demand nodes of non-zero capacity. As established in Section 5.1, we may resolve as much flow as possible through u_1, since we would gain no advantage by deferring either flow to a more distant supply-demand pair. Therefore, we may route $\min(s_1', d_1')$ units of internal flow through u_1. Similarly, we may route $\min(s_k', d_k')$ units of internal flow through u_k. These flows are added to I. Let (s_1'', d_1'') denote the remaining supply/demand capacities for u_1 (only one of these will be nonzero), and let (s_k'', d_k'') be the analogous values for u_k (see Fig. 17(c)). This completes the collapsing process for forward monotone subchains. Backward subchains are processed symmetrically. We show that network flows are preserved, up to the addition of the internal flows.

Lemma 10. *Consider a monotone subchain $U = \langle u_1, \ldots, u_k\rangle$ in G (after subtree collapsing). Let G' denote the network after collapsing this subchain, and let $I(U)$ denote the internal flow as generated by Algorithm 2. Then for any flow f in G, there exists in G' a flow of value at least $|f| - I(U)$, and given any flow f' in G', there exists in G a flow of value at least $|f'| + I(U)$. This latter flow can be computed in time proportional to the size of the chain.*

Proof. The proof of the second assertion follows directly from the remarks made during our description of the collapsing process. In particular, each step of the collapsing process involves identifying a path in U along which internal flow can be routed, and reducing the supply and demand capacities accordingly. By simply reversing the process, we can easily map any flow f' in G' to a flow f'' in G. The restoration of the internal flows to f'' increases its flow value by $I(U)$, and we let f denote the result. Clearly, the algorithm runs in time proportional to the size of the each chain.

The proof of the first assertion involves showing first that any flow in G can be modified to a flow of equivalent value that satisfies the general structural properties of the flow generated by the collapsing process. We show this first for the partially collapsed network. From Lemma 9, we may assume that f's internal flow within the core matches the internal flow generated by Algorithm 2, and additionally, it satisfies up to D' units of demand and S' units of supply within the core. As observed earlier, the flow paths of f that terminate at demand nodes in the core must flow through u_1, and so we may reroute these paths directly into d_1, which is legal because the capacity of d_1 has been increased by D' units in the partially collapsed network. Similarly, the flow paths of f that originate at supply nodes in the core must flow through u_k, and we may reroute these paths directly into s_k, which is legal because the capacity of s_k has been increased by S' units in the partially collapsed network.

Therefore, within this subchain, we may assume that f has been transformed into a flow that is valid for the partially collapsed network and whose internal flow equals the internal flow of Algorithm 2. To complete the proof, we observe that the correctness of the final collapsing step (merging the supply and demands of u_1 and u_k) follows from Lemma 8. □

By applying the above lemma to every monotone subchain within G, we obtain the following.

Lemma 11. *Let G be a network (after applying the subtree collapsing process), and let G' be the network resulting after applying the collapsing process on all the monotone subchains of G. Let $I_M(G)$ denote the total internal flow generated in the process. Then for any flow f in G, there exists in G' a flow of value at least $|f| - I_M(G)$, and given any flow f' in G', there exists in G a flow of value at least $|f'| + I_M(G)$. This latter flow can be computed in $O(n)$ time, where n is the size of G.*

5.3. Collapsing zig–zag chains

Next, we show how to process collapsing zig–zag chains. Consider a maximal chain $U = \langle u_1, \ldots, u_k\rangle$ of degree-two nodes in the network after collapsing monotone subchains. Since each maximal monotone subchain has been replaced by a single edge, this is a zig–zag chain (see Fig. 18(a)), which alternates between forward and backward edges. We are interested in collapsing arbitrarily long chains to chains of a constant size, and so we may assume that $k \geq 3$, and by removing the first node and/or last node of the chain if necessary, we may assume that the entry and exit edges are both forward edges. We may assume that each even numbered node u_i has no net demand (since such demand could not be reached by any supply), and symmetrically, each odd numbered node u_i has no net supply. Thus, we may assume that each even numbered node is incident only to a supply node s_i, and each odd numbered node is incident only to a demand node d_i. (If such a node does not exist, we create a trivial supply or demand node of capacity 0.) We say that such a chain is *admissible*.

Our approach is to treat the flow on the entry edge as a variable, f_{in}, and derive a flow, which is based on this assumption about the entry flow. Our flow satisfies demands and supplies in a greedy manner. The flow along the ith edge of the chain will be denoted by f_i. The chain's entry edge is the 0th edge, so $f_0 = f_{in}$. In order to motivate our approach, first observe that, since u_1 is incident to a demand node of capacity d_1, we may assume that $0 \leq f_{in} \leq d_1$. Up to d_1 units of demand are satisfied, which leaves up to $d_1 - f_{in}$ units of flow that may be pushed along edge (u_2, u_1). Since u_2 can generate up to s_2 units of this supply, the maximum flow that can be pushed along this edge is $f_1 = \min(s_2, d_1 - f_{in})$. It is easy to see that there is no benefit to be gained by pushing less flow along this edge, since in order to achieve the same total flow we would need to push more flow on subsequent edges, and this might violate subsequent capacity constraints. By applying the same reasoning to the next edge of the chain, we see that the flow

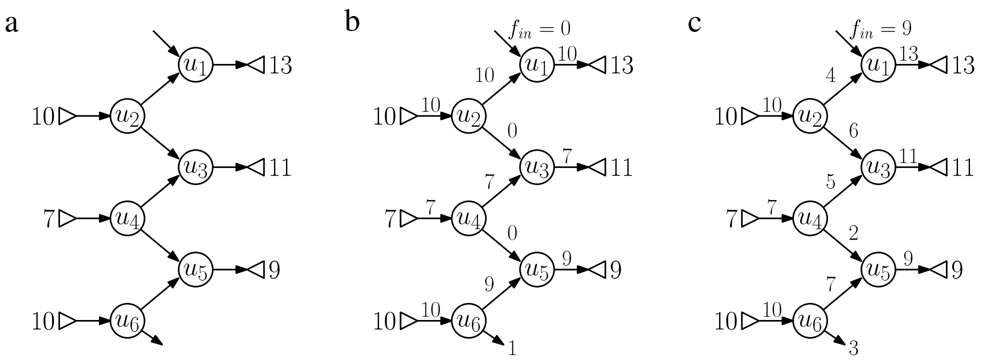


Fig. 18. Flows in a zig–zag chain: (a) a chain of length $k = 6$, and (b)–(c) examples of greedy flows for $f_{in} = 0$ and 9.

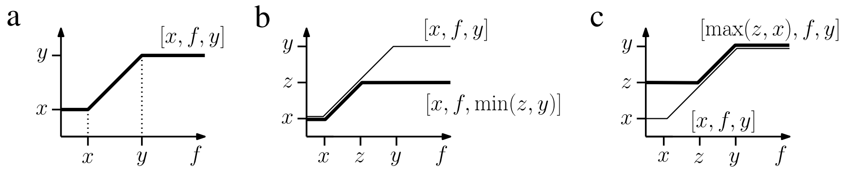


Fig. 19. An illustration of the function $[x, f, y] = \max(x, \min(f, y))$ and Lemma 12.

along (u_2, u_3) is at most $f_2 = \min(d_3, s_2 - f_1)$. (Note that $f_1 \le s_2$, and therefore f_2 is nonnegative.) Continuing in this manner, we can compute the maximum flow along each edge of the chain, subject to the assumption that the incoming flow is f_{in} and our principle of greedily applying flow (see Fig. 18(b) and (c) for two examples). Although it is tempting to think that the complexity of any formula describing this flow would grow as some function of k, we will show that it is possible to express each of these flows by a formula of constant size. Furthermore, we can also produce a formula of constant size for the *total internal flow* in the chain, that is, the sums of flows between the nodes of the chain. This will enable us to replace the entire chain by a single flow-equivalent chain of constant size.

Since we will be manipulating expressions involving max and min, it will simplify the presentation to define some helpful notation. Given values $x \le y$, define $[x, f, y]$ to be a shorthand for $\max(x, \min(f, y))$. Intuitively, this "clamps" the value of f between x and y (see Fig. 19(a)). The next lemma summarizes a few important facts about this notation.

Lemma 12. *Given reals x, y, and z:*

(i) *if* $x \le \min(y, z)$, *then* $\min(z, [x, f, y]) = [x, f, \min(y, z)]$ *(see Fig. 19 (b))*
(ii) *if* $\max(x, z) \le y$, *then* $\max(z, [x, f, y]) = [\max(x, z), f, y]$ *(see Fig. 19 (c))*
(iii) *if* $x \le y$, *then* $[x, f, y] = x + [0, f - x, y - x]$
(iv) *if* $0 \le y \le z$, *then* $[0, f - x, z] - [0, f - x, y] = [0, f - (x + y), z - y]$
(v) *if* $0 \le y \le z$, *then* $[0, f - (x + y), z - y] + [0, f - x, y] = [0, f - x, z]$.

Proof. Claims (i)–(iii) are easy consequences of the definition of the notation. To prove claim (iv), we consider two cases. If $f - x \le y$, then, since $y \le z$, both $[0, f - x, z]$ and $[0, f - x, y]$ are equal to $\max(0, f - x)$, and therefore their difference is 0. In this case, $f - (x + y) \le 0$, and so $[0, f - (x + y), z - y] = 0$. On the other hand, if $f - x > y$, then $[0, f - x, z] = \min(f - x, z)$ and $[0, f - x, y] = y$. Thus, their difference is $\min(f - x, z) - y = \min(f - (x + y), z - y)$. Observe that this quantity is nonnegative when $f - x > y$. Combining these two cases, we have $[0, f - x, z] - [0, f - x, y] = [0, f - (x + y), z - y]$. Finally, claim (v) follows easily from (iv). □

Given an (unknown) flow of f_{in} on the chain's entry edge, we will show how to compute quantities c_i, b_i, and r_i such that there exists a flow of the form $f_i = c_i + [0, f_{in} - r_{i-1}, b_i]$ along each forward edge (u_{i-1}, u_i) and of value $f_i = c_i - [0, f_{in} - r_i, b_i]$ along each backward edge (u_i, u_{i-1}). Intuitively, c_i indicates the *base capacity*, which is the amount of flow on the edge if $f_{in} = 0$. The quantity b_i indicates the *bottleneck*, which is the maximum amount by which the flow on this edge may change, depending on variations in f_{in}. Finally, r_i denotes the *residual demand*, which indicates how large f_{in} must be in order to affect the flow on this edge. In order to compute the total internal flow, we compute a fourth quantity t_i, which reflects the total internal flow for the first i edges when $f_{in} = 0$.

Algorithm 3 shows how these quantities are computed given the chain's supply and demand capacities as input. While the formulas given in the algorithm are not complicated, it is not easy to justify them intuitively. Their correctness is established below in Lemma 14. (It follows from the lemma's proof that all the quantities take on nonnegative integer values.)

Algorithm 3: chain-flow(U), computing the flow coefficients along an admissible zig–zag chain $U = \langle u_1, \ldots, u_k \rangle$.

```
$(c_0, r_{-1}, r_0, b_0, t_0) \leftarrow (0, 0, d_1, d_1, 0)$;          // initialize
for $i \leftarrow 1$ to $k$ do      // process each node of the chain
    if ($i$ is odd) then        // $u_i$ is incident to demand $d_i$
        $\Delta_i \leftarrow d_i - c_{i-1}$;                   $c_i \leftarrow \min(\Delta_i, s_{i+1})$;
        $b_i \leftarrow b_{i-1} - [0, \Delta_i - s_{i+1}, b_{i-1}]$;   $r_i \leftarrow r_{i-1} - b_i$;
    else                        // $u_i$ is incident to supply $s_i$
        $\Delta_i \leftarrow s_i - c_{i-1}$;                   $c_i \leftarrow \min(\Delta_i, d_{i+1})$;
        $b_i \leftarrow [0, d_{i+1} - \Delta_i, b_{i-1}]$;             $r_i \leftarrow r_{i-1} + b_i$;
    $t_i \leftarrow t_{i-1} + c_i$;
return $(c_k, b_k, r_{k-1}, t_{k-1})$;     // parameters defining the
exit and internal flows
```

Fig. 20(a) shows the quantities computed by Algorithm 3 for the chain of Fig. 18(a). Given the returned values (c, b, r, t), define $f_{out}(c, b, r) = c + [0, f_{in} - r, b]$ and $I(r, t) = t - [0, f_{in} - r, d_1 - r]$. Fig. 20(b) shows the flow formulas for each edge. The algorithm returns $(c, b, r, t) = (1, 6, 7, 26)$. The exit flow is $f_{out}(c, b, r) = 1 + [0, f_{in} - 7, 6]$, and the total internal flow is $I(r, t) = 26 - [0, f_{in} - 7, 6]$.

a

i	Δ_i	c_i	b_i	r_i	t_i
0	–	0	13	13	0
1	13	10	10	3	10
2	0	0	10	13	10
3	11	7	6	7	17
4	0	0	6	13	17
5	9	9	6	**7**	**26**
6	1	**1**	**6**	–	–

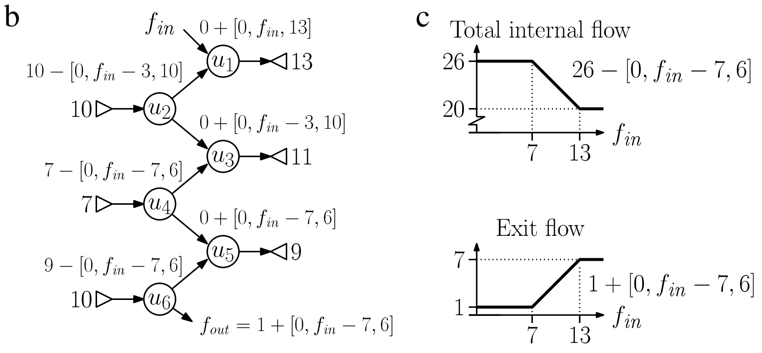


Fig. 20. Example of Algorithm 3: (a) computed quantities for the chain of Fig. 18(a), (b) the flow functions for each edge, (c) the exit flow and total internal flow functions.

Before establishing the algorithm's correctness, we present a technical lemma regarding the quantities computed by the algorithm.

Lemma 13. *Given an admissible zig–zag chain $U = \langle u_1, \dots, u_k\rangle$, the quantities computed by Algorithm 3 satisfy the following conditions for $0 \le i \le k$:*

(i) *if $i \ge 1$, $0 \le b_i \le b_{i-1}$*
(ii) *if i is odd, $0 \le r_i \le r_i + b_i \le d_1$*
(iii) *if i is even, $0 \le r_i \le d_1$.*

Proof. To prove (i), observe that, by definition of the bracket notation, both $[0, \Delta_i - s_{i+1}, b_{i-1}]$ and $[0, d_{i+1} - \Delta_i, b_{i-1}]$ are nonnegative and at most b_{i-1}. It follows directly (in either the even or odd case) that $0 \le b_i \le b_{i-1}$, as desired.

Assertions (ii) and (iv) are established by induction. For the basis cases ($i \in \{-1, 0\}$), by definition, $r_{-1} = 0$ and $r_0 = d_1$, so $0 \le r_{-1} \le r_0 \le d_1$. If $i > 0$ is odd, then $r_i + b_i = (r_{i-1} - b_i) + b_i = r_{i-1}$, and by induction $r_{i-1} \le d_1$. Also $r_i = r_{i-1} - b_i = r_{i-2} + b_{i-1} - b_i$. Since $b_i \le b_{i-1}$, by induction we have $r_i \ge r_{i-2} \ge 0$. On the other hand, if $i > 0$ is even, then $r_i = r_{i-1} + b_i \le r_{i-1} + b_{i-1}$, which by induction is at most d_1. Since $b_i \ge 0$, we have $r_i \ge r_{i-1}$. By induction, $r_{i-1} \ge 0$, and therefore, $0 \le r_i \le d_1$. □

The following lemma summarizes the nature of the flow computed by Algorithm 3.

Lemma 14. *Given an admissible zig–zag chain $U = \langle u_1, \dots, u_k\rangle$, let (c, b, r, t) be the quantities returned by Algorithm 3. For any real f_{in}, where $0 \le f_{in} \le d_1$, there exists a flow through U whose entry edge carries flow f_{in}, whose exit edge carries flow $f_{out}(c, b, r)$, and whose total internal flow is $I(r, t)$.*

Proof. Given the values computed by Algorithm 3, we will prove that, for $0 \le i \le k$, there is a flow such that, if i is odd, the ith edge carries flow of $f_i = c_i - [0, f_{in} - r_i, b_i]$, and if i is even it carries flow $f_i = c_i + [0, f_{in} - r_{i-1}, b_i]$. (Recall that edge 0 is the entry edge.) By Lemma 13(i), b_i is nonnegative, and so these expressions satisfy the requirements of our bracket notion. We will also show that the capacity constraints are satisfied. In particular, if i is odd, then $0 \le f_i \le \min(d_i, s_{i+1})$, and if i is even, then $0 \le f_i \le \min(s_i, d_{i+1})$. Finally, let I_i denote the sum of flows on the first i edges of the chain, not counting the entry edge. That is, $I_i = \sum_{j=1}^{i} f_j$. We will show that $I_i = t_i - [0, f_{in} - r_i, d_1 - r_i]$.

To see that this suffices to establish the lemma, recall that the chain has an even number of nodes, and so the output flow is $f_k = c_k + [0, f_{in} - r_{k-1}, b_k] = c + [0, f_{in} - r, b] = f_{out}(c, b, r)$. Because the total internal flow is determined by the flow on edges 1 through $k-1$, we have $I_{k-1} = t_{k-1} - [0, f_{in} - r_{k-1}, d_1 - r_{k-1}] = t - [0, f_{in} - r, d_1 - r] = I(r, t)$.

For the basis, observe that for $i = 0$, the flow is $f_0 = f_{in}$. Since u_1 can handle at most d_1 units of flow, we may assume that $0 \le f_{in} \le d_1$, which can be equivalently expressed as $f_0 = 0 + [0, f_{in} + 0, d_1] = c_0 + [0, f_{in} - r_{-1}, b_0]$. The internal flow is zero, which can be expressed as $I_0 = 0 - [0, f_{in} - d_1, 0] = 0 - [0, f_{in} - r_0, d_1 - r_0]$, as desired.

For the induction step, first consider the case where i is odd, which implies that u_i is adjacent to the demand node d_i (see Fig. 21(a)). By the induction hypothesis, the flow on the incoming edge to u_i is $f_{i-1} = c_{i-1} + [0, f_{in} - r_{i-2}, b_{i-1}]$ and $0 \le f_{i-1} \le \min(s_{i-1}, d_i)$. Up to d_i units of this flow may be absorbed, leaving $d_i - f_{i-1}$ units of flow for edge (u_{i+1}, u_i). Up to s_{i+1} units of supply can be provided by the subsequent supply node. Thus, we may assign $\min(s_{i+1}, d_i - f_{i-1})$ units of flow to this edge. Letting $\Delta_i = d_i - c_{i-1}$, we have

$$\begin{aligned} f_i &= \min(s_{i+1}, d_i - f_{i-1}) \\ &= \min(s_{i+1}, d_i - (c_{i-1} + [0, f_{in} - r_{i-2}, b_{i-1}])) \\ &= \min(s_{i+1}, \Delta_i - [0, f_{in} - r_{i-2}, b_{i-1}]) \\ &= \Delta_i - \max(\Delta_i - s_{i+1}, [0, f_{in} - r_{i-2}, b_{i-1}]), \end{aligned}$$

where in the last step we used the fact that $\min(x, y - z) = y - \max(y - x, z)$.

We consider two subcases. First, suppose that $\Delta_i - s_{i+1} \le b_{i-1}$. Let $x = 0$, $y = b_{i-1}$, and $z = \Delta_i - s_{i+1}$. By Lemma 13(i) we have $0 \le b_{i-1}$, and so for this subcase we have $\max(x, z) \le y$. Thus, we satisfy the preconditions of Lemma 12(ii), from which we have

$$f_i = \Delta_i - [\max(0, \Delta_i - s_{i+1}), f_{in} - r_{i-2}, b_{i-1}].$$

Setting $\Gamma = \max(0, \Delta_i - s_{i+1})$, then by Lemma 12(iii) (with $x = \Gamma$ and $y = b_{i-1}$) we have,

$$\begin{aligned} f_i &= \Delta_i - [\Gamma, f_{in} - r_{i-2}, b_{i-1}] \\ &= (\Delta_i - \Gamma) - [0, f_{in} - r_{i-2} - \Gamma, b_{i-1} - \Gamma]. \end{aligned}$$

To simplify the first term, observe that $\Gamma = \max(0, \Delta_i - s_{i+1}) = \Delta_i - \min(\Delta_i, s_{i+1}) = \Delta_i - c_i$, and therefore, $\Delta_i - \Gamma = c_i$. Note that in this subcase, we can express Γ as $[0, \Delta_i - s_{i+1}, b_{i-1}]$. Thus, by definition of b_i, we have $b_{i-1} - \Gamma = b_i$. By definition of r_{i-1} (even case) and r_i (odd case), we have $r_{i-1} = r_{i-2} + b_{i-1}$ and $r_i = r_{i-1} - b_i$. Therefore, we obtain

$$\begin{aligned} r_{i-2} + \Gamma &= (r_{i-1} - b_{i-1}) + \Gamma = r_{i-1} - (b_{i-1} - \Gamma) \\ &= r_{i-1} - b_i = r_i. \end{aligned}$$

Combining these observations, we have $f_i = c_i - [0, f_{in} - r_i, b_i]$, as desired.

The second subcase is $\Delta_i - s_{i+1} > b_{i-1}$. In this case we have

$$\begin{aligned} f_i &= \Delta_i - \max(\Delta_i - s_{i+1}, [0, f_{in} - r_{i-2}, b_{i-1}]) \\ &= \Delta_i - (\Delta_i - s_{i+1}) = s_{i+1}. \end{aligned}$$

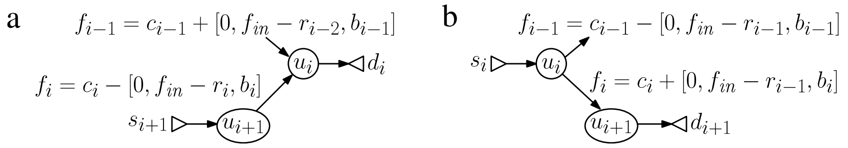

a
$f_{i-1} = c_{i-1} + [0, f_{in} - r_{i-2}, b_{i-1}]$
$f_i = c_i - [0, f_{in} - r_i, b_i]$
u_i
d_i
s_{i+1}
u_{i+1}
b
$f_{i-1} = c_{i-1} - [0, f_{in} - r_{i-1}, b_{i-1}]$
s_i
u_i
$f_i = c_i + [0, f_{in} - r_{i-1}, b_i]$
u_{i+1}
d_{i+1}

Fig. 21. Proof of Lemma 14: (a) for odd i and (b) for even i.

This can be expressed equivalently as $f_i = s_{i+1} - [0, f - r_i, 0]$. By Lemma 13(i) $b_{i-1} \geq 0$, and so in this subcase $\Delta_i > s_{i+1}$. Therefore, $c_i = \min(\Delta_i, s_{i+1}) = s_{i+1}$. By definition, $b_i = b_{i-1} - [0, \Delta_i - s_{i+1}, b_{i-1}] = b_{i-1} - b_{i-1} = 0$. Therefore, we have $f_i = c_i - [0, f_{in} - r_i, b_i]$, as desired.

Let us also consider the total internal flow when i is odd. By the definition of internal flow and the induction hypothesis we have

$$\begin{aligned} I_i &= I_{i-1} + f_i \\ &= (t_{i-1} - [0, f_{in} - r_{i-1}, d_1 - r_{i-1}]) + (c_i - [0, f_{in} - r_i, b_i]) \\ &= (t_{i-1} + c_i) - ([0, f_{in} - r_{i-1}, d_1 - r_{i-1}] + [0, f_{in} - r_i, b_i]). \end{aligned}$$

By definition, $t_{i-1} + c_i = t_i$ and (since i is odd) $r_i + b_i = r_{i-1}$, which yields

$$I_i = t_i - ([0, f_{in} - (r_i + b_i), d_1 - r_i - b_i] + [0, f_{in} - r_i, b_i]).$$

By setting $x = r_i$, $y = b_i$, and $z = d_1 - r_i$, it follows from Lemma 13 that the preconditions of Lemma 12(v) hold, and by applying this lemma we have $I_i = t_i - [0, f_{in} - r_i, d_1 - r_i]$, as desired.

Next, we consider the case where i is even, which implies that u_i is adjacent to the supply node s_i (see Fig. 21(b)). By the induction hypothesis, the flow on the incoming edge to u_i is $f_{i-1} = c_{i-1} - [0, f_{in} - r_{i-1}, b_{i-1}]$ and $0 \leq f_{i-1} \leq \min(d_{i-1}, s_i)$. Up to s_i units of this flow may be supplied by s_i, leaving $s_i - f_{i-1}$ units of flow for edge (u_i, u_{i+1}). Up to d_{i+1} units of flow can be absorbed by the subsequent demand node. Thus, we may assign $\min(d_{i+1}, s_i - f_{i-1})$ units of flow to this edge. Letting $\Delta_i = s_i - c_{i-1}$, we have

$$\begin{aligned} f_i &= \min(d_{i+1}, s_i - f_{i-1}) \\ &= \min(d_{i+1}, s_i - (c_{i-1} - [0, f_{in} - r_{i-1}, b_{i-1}])) \\ &= \min(d_{i+1}, \Delta_i + [0, f_{in} - r_{i-1}, b_{i-1}]) \\ &= \Delta_i + \min(d_{i+1} - \Delta_i, [0, f_{in} - r_{i-1}, b_{i-1}]). \end{aligned}$$

As before, we consider two subcases. If $d_{i+1} - \Delta_i \geq 0$, then $c_i = \Delta_i$, and $\min(b_{i-1}, d_{i+1} - \Delta_i)$ can be expressed as $[0, d_{i+1} - \Delta_i, b_{i-1}]$, which by definition is b_i. Let $x = 0$, $y = b_{i-1}$, and $z = d_{i+1} - \Delta_i$. By Lemma 13(i) we have $0 \leq b_{i-1}$, and so for this subcase we have $x \leq \min(y, z)$. Thus, we satisfy the preconditions of Lemma 12(i), from which we have

$$\begin{aligned} f_i &= \Delta_i + [0, f_{in} - r_{i-1}, \min(b_{i-1}, d_{i+1} - \Delta_i)] \\ &= c_i + [0, f_{in} - r_{i-1}, b_i], \end{aligned}$$

as desired.

The second subcase is $d_{i+1} - \Delta_i < 0$. Clearly, $f_i = \Delta_i + (d_{i+1} - \Delta_i) = d_{i+1}$, which can be expressed equivalently as $d_{i+1} + [0, f - r_{i-1}, 0]$. In this case we have $c_i = d_{i+1}$, and since $b_i = [0, d_{i+1} - \Delta_i, b_{i-1}]$, we have $b_i = 0$. Therefore, $f_i = c_i + [0, f_{in} - r_{i-1}, b_i]$, as desired.

Finally, let us consider the total internal flow when i is even. Clearly,

$$\begin{aligned} I_i &= I_{i-1} + f_i \\ &= (t_{i-1} - [0, f_{in} - r_{i-1}, d_1 - r_{i-1}]) + (c_i + [0, f_{in} - r_{i-1}, b_i]). \end{aligned}$$

By definition, $t_{i-1} + c_i = t_i$ and (since i is even) $r_{i-1} = r_i - b_i$, which yields

$$I_i = t_i - ([0, f_{in} - r_{i-1}, d_1 - r_i + b_i] - [0, f_{in} - r_{i-1}, b_i]).$$

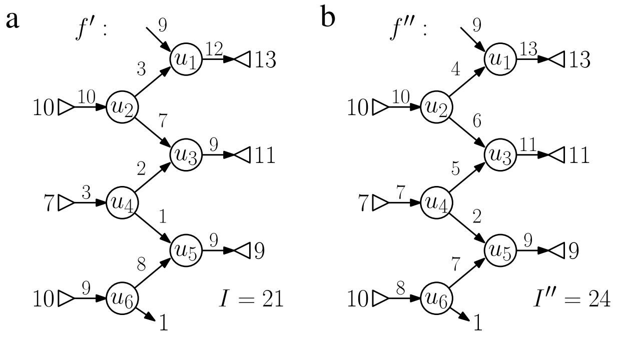


Fig. 22. Proof of Lemma 15.

By setting $x = r_{i-1}$, $y = b_i$, and $z = d_1 - r_i + b_i$, it follows from Lemma 13(iii) that the preconditions of Lemma 12(iv) hold, and so by applying this lemma we have

$$\begin{aligned} I_i &= t_i - [0, f_{in} - (r_{i-1} + b_i), (d_1 - r_i + b_i) - b_i] \\ &= t_i - [0, f_{in} - r_i, d_1 - r_i], \end{aligned}$$

which completes the case where i is even, and so completes the proof. □

The previous lemma shows the existence of a flow through a zig–zag chain U. Next, we show that this is essentially the best possible, in the sense that given any valid flow through U, such that the entry edge carries flow f_{in}, there exists a flow of greater or equal value that has the same structure as the flow described in the previous lemma.

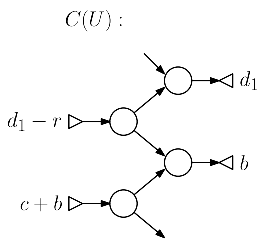
Lemma 15. *Consider an admissible zig–zag chain $U = \langle u_1, \ldots, u_k \rangle$ in a network G. Let (c, b, r, t) be the quantities returned by Algorithm 3. Let f' be any valid flow in G and let f'_{in}, f'_{out}, and I', denote the entry, exit, and total internal flow for U, respectively. Then, there exists a flow f'' in G of equal or greater value to f', whose entry flow is f'_{in}, whose exit flow is not greater than f'_{out}, and whose total internal flow within U is $I(r, t)$.*

Proof. Consider any legal flow f' through the chain (see Fig. 22(a)). We construct the flow f'' as follows. The entry edge carries f'_{in}. For $1 \leq i \leq k$, let f''_i denote the value of flow on the ith edge as given in the proof of Lemma 14, under the assumption that the entry flow is f'_{in}. That is, $f''_i = c_i - [0, f'_{in} - r_i, b_i]$ if i is odd and $f''_i = c_i + [0, f'_{in} - r_{i-1}, b_i]$, if i is even. Finally, the exit edge carries the minimum of f'_{out} and the exit flow $f_{out}(c, b, r)$, defined earlier (see Fig. 22(b)).

We assert that f'' is a valid flow. Except for the exit edge, all edges carry the same flow as computed in the proof of Lemma 14, and the flow on the exit edge is no larger than the exit flow established in the lemma. Therefore, all the supply and demand capacity constraints can be satisfied.

To complete the proof, we show that the total value of f'' is at least as large as that of f'. We do this by successively modifying f' in a manner that does not decrease its total value, until we obtain a new flow that is equal to f''.

Assuming that the two flows differ, consider the first edge (u_i, u_{i+1}) on which they differ. Since f' and f'' have the same entry flow, we have $1 \leq i \leq k$. Let f'_i and f''_i denote the flows of f' and a



b

i	Δ_i	c_i	b_i	r_i	t_i
0	–	0	d_1	d_1	0
1	d_1	$d_1 - r$	$d_1 - r$	r	$d_1 - r$
2	0	0	b	$b + r$	$d_1 - r$
3	b	b	b	$\mathbf{r}$	$\mathbf{d_1 - r + b}$
4	c	$\mathbf{c}$	$\mathbf{b}$	–	–

c

$$f_{\text{out}} = c - [0, f_{\text{in}} - r, b]$$

$$I = (d_1 - r + b) - [0, f_{\text{in}} - r, d_1 - r]$$

Fig. 23. Zig–zag chain collapsing: (a) the collapsed structure $C(U)$ for a network U whose first demand capacity is d_1 and for which (c, b, r, t) is the output of Algorithm 3, (b) a summary of the computations of Algorithm 3, (c) the exit flow and total internal flow.

f'' on $e = (u_i, u_{i+1})$, respectively. As in Lemma 14, f''_i is chosen to be the maximum flow on e, subject to the incoming flow (which is the same for both flows) and the capacity constraints associated with u_i and u_{i+1}. Therefore, $f'_i < f''_i$. This implies that $i < k$, since by definition, $f''_k \le f'_{out} = f'_k$. Let $\Delta = f''_i - f'_i$. We modify f' by increasing the flow on e by Δ and decreasing the flow on the next edge of the chain by $\min(f'_{i+1}, \Delta)$. The resulting flow is clearly valid, and this modification does not decrease the total internal flow of f'. Eventually, the two flows will be identical, which establishes that the total internal flow of f'' is at least as large as that of f'. □

The above lemma implies that, for each admissible zig–zag chain, the characterization given in Lemma 14 of the flow produced by Algorithm 3 is essentially complete, since, by applying this to every chain of the network, we may assume that any maximum flow is of this form. As mentioned earlier, our approach is to collapse each such zig–zag chain in the network into a zig–zag chain of constant size such that Algorithm 3 returns the same values on the replacement chain as for the original chain. By the above lemma, the two chains are flow-equivalent.

Given an admissible zig–zag chain $U = \langle u_1, \ldots, u_k\rangle$, let (c, b, r, t) be the quantities returned from Algorithm 3. Before presenting the replacement network, we give a lemma that establishes some useful relations among these quantities.

Lemma 16. *Given an admissible zig–zag chain $U = \langle u_1, \ldots, u_k\rangle$, the quantities (c, b, r, t) returned by Algorithm 3 are all nonnegative. Also, recalling that d_1 is the demand capacity associated with u_1, the quantities b and r satisfy $b \le d_1$ and $r \le b + r \le d_1$.*

Proof. By definition, $(c, b, r, t) = (c_k, b_k, r_{k-1}, t_{k-1})$. The proof of Lemma 14 establishes that c_k is the flow on the exit edge for a valid flow (when $f_{in} = 0$) and t_{k-1} is the sum of edge flow in the same flow. Therefore, c_k and t_{k-1} are both nonnegative. Recalling that k is even, the nonnegativity of b_k and r_{k-1} follows from Lemma 13(i) and (iii). Lemma 13(i) implies that the b_i's are nonincreasing, and since $b_0 = d_1$, we have $b = b_k \le d_1$. Since $k - 1$ is odd, by Lemma 13(ii) we have $r_{k-1} \le r_{k-1} + b_{k-1} \le d_1$. Since $b_{k-1} \le b_k$, we have $r \le r + b \le d_1$, which completes the proof. □

Given U and the quantities (c, b, r, t) returned from Algorithm 3, the collapsed zig–zag network, denoted $C(U)$, is shown in Fig. 23(a). By the above lemma, all the capacities are nonnegative, so this is a valid chain. Our next lemma together with Lemma 15 implies that, with respect to maximum flows, the replacement chain is equivalent to the original chain.

Lemma 17. *Consider an admissible zig–zag chain $U = \langle u_1, \ldots, u_k\rangle$. Let (c, b, r, t) denote the quantities returned by Algorithm 3 on U. Then, when run on $C(U)$, this algorithm returns $(c, b, r, d_1 - r + b)$.*

Proof. The proof involves running the algorithm on $C(U)$. A summary of the execution of this algorithm is shown in Fig. 23(b), where the returned values are shown in bold. The only nontrivial quantities to compute are the b_i's. Their values are justified below.

$i = 1$: $b_1 = b_0 - [0, \Delta_1 - s_2, b_0] = d_1 - [0, r, d_1]$. By Lemma 16, $0 \le r \le d_1$, so $b_1 = d_1 - r$.
$i = 2$: $b_2 = [0, d_3 - \Delta_2, b_1] = [0, b, d_1 - r]$. By Lemma 16, $0 \le b \le d_1 - r$, so $b_2 = b$.
$i = 3$: $b_3 = b_2 - [0, \Delta_3 - s_4, b_2] = b - [0, b - (c + b), b] = b - 0 = b$.
$i = 4$: $b_4 = [0, d_5 - \Delta_4, b_3] = [0, \infty, b] = b$ (by our convention that $d_{k+1} = \infty$). □

Together, Lemmas 14, 15 and 17 imply that, if we collapse an admissible zig–zag chain U by replacing it with $C(U)$, we obtain a chain of constant size that differs from U only with respect to the fact that it generates $t - (d_1 - r + b)$ fewer units of internal flow, which we denote by $I(U)$. (It can be shown that this quantity is always nonnegative, but the correctness of our results does not depend on this.) By collapsing all the admissible zig–zag chains of G, we obtain the main result of this section.

Lemma 18. *Let G be a network (after subtree and monotone chain collapsing), and let G' be the result of applying the zig–zag chain collapsing process to G. Let $I_Z(G)$ denote the sum of $I(U)$ over all admissible zig–zag chains in G. Given a flow f in G, there exists of flow in G' of value at least $|f| - I_Z(G)$. Conversely, given any flow f' in G', there exists a flow in G of value at least $|f'| + I_Z(G)$. This latter flow can be computed in $O(n)$ time, where n is the size of G.*

5.4. Maximum flow in the collapsed network

In this section we combine the results of the previous three sections. Let $G = (V, E)$ denote a planar multi-supply, multi-demand network, as defined in Section 4. Let $G' = (V', E')$ denote the network resulting after collapsing subtrees, monotone chains, and zig–zag chains. Recall the internal flow values $I_T(G)$, $I_M(G)$ and $I_Z(G)$ defined in Lemmas 8, 11 and 18, respectively. Combining these lemmas we have the following.

Lemma 19. *Let G be a planar multi-supply, multi-demand network, and let G' be the result of applying the subtree, monotone chain, and zig–zag chain collapsing processes to G. Let $I(G) = I_T(G) + I_M(G) + I_Z(G)$. Given a flow f in G, there exists of flow in G' of value at least $|f| - I(G)$. Conversely, given any flow f' in G', there exists a flow in G of value at least $|f'| + I(G)$. This latter flow can be computed in $O(n)$ time, where n is the size of G.*

Thus, computing the maximum flow in G reduces to computing the maximum flow in the collapsed network G' and then adding back the internal flows $I(G)$, which were lost in the collapsing process. The running time of the entire procedure is equal to sum of the time to produce the collapsed network, the time to compute the maximum flow in G', and the time to convert the flow in G' to a flow in G. It follows from the remarks made in earlier sections that the collapsed network can be computed in $O(n)$ time, and, by the above lemma, the time to convert the flow in G' to a flow in G is $O(n)$. It only remains to analyze the size of the collapsed network and the time to compute the maximum flow in this network.

Lemma 20. *The collapsed network G' is of size $O(h)$, where h is the number of faces of G.*

Proof. For now, let us ignore the supply and demand nodes and also the edge directions of G'. The result is a planar graph. The collapsing process does not remove cycles from the graph, and therefore G' has h faces. After collapsing, each chain consists of a constant number of vertices and edges. By replacing each chain by a single edge, the total size of the graph is within a constant factor of the original. Since all chains and tree-like structures have been removed, every vertex of the resulting planar graph is incident to three or more edges.

Let v, e, and h denote the numbers of vertices, edges, and faces in this graph. By summing the degrees of the vertices, we count each edge twice, and since all the vertices are of degree at least three, we have $2e = \sum_u \deg(u) \geq 3v$. Since the graph is planar and connected, by Euler's formula [17] we have $v - e + h = 2$. Therefore, $v-(3/2)v+h \geq 2$, or equivalently $v \leq 2(h-2) = O(h)$. Also, the number of edges is $e = v + h - 2 \leq 3(h-2) = O(h)$. Therefore, the total size of G' is $O(h)$. □

In order to compute the maximum flow in G', we apply a recent result due to Borradaile et al. [15], which shows that the maximum flow in a planar multiple source, multiple sink network of size n can be computed in $O(n \log^3 n)$ time. In our case, the size of G' is $O(h)$, and so the time to compute the maximum flow is $O(h \log^3 h)$. The algorithm of Borradaile et al. assumes that capacities are stored on edges, while our network has node capacities. Because only the supply and demand nodes have finite capacities, we may simply move each such node's capacity to its (unique) incident edge. Let f' be the resulting maximum flow in G'. By Lemma 19, in $O(n)$ time, the resulting flow can be mapped back to a flow f in the original network G. In summary we have the following.

Theorem 1. *Given a planar n-vertex network G with multiple sources and multiple sinks and h faces. It is possible to compute a maximum flow in G in time $O(n + h \log^3 h)$ time.*

Let us now apply this result to solve the MUMC problem. Recall that the input consists of a simple n-vertex polygon P with h holes. Let R denote the number of reflex vertices in P, and let r denote the number of scan reflex vertices (SRVs). In $O(n \log n)$ time we can compute the pseudo-trapezoidal decomposition of P. In $O(n)$ time we can compute the associated flow network G, which was introduced at the start of Section 4. As shown in Lemmas 1, 3 and 4, the maximum flow in G, denoted $|f|$, determines the number of monotone components in an x-monotone subdivision. In particular, by Lemmas 3 and 4, the number of monotone components is $k = (r - h + 1) - |f|$. Let G' denote the collapsed network. The collapsing process does not remove cycles from the graph, and therefore G' has $h + 1$ faces, one for each hole of P and one for P's exterior. By Theorem 1, we can compute this maximum flow in time $O(n + h \log^3 h)$.

The overall running time is $O(n \log n + h \log^3 h)$. As shown in Section 4, the entire subdivision can be output in time proportional to its size, which is $O(n + R^2)$. By Lemma 5, we can store the flow in a manner that uses only $O(n)$ space and allows us to extract the boundary of any x-monotone component in time proportional to its size, which is $O(n + R)$.

This provides a solution to the standard MUMC problem where the monotonicity direction is specified. To solve the general MUMC problem (where the monotonicity direction is not specified), we apply a result of Arkin et al. [14], which shows that it suffices to test $O(K)$ reference directions, where K is the number of edges of the polygon's visibility graph. In summary, we have our main result.

Theorem 2. *Consider an n-vertex polygon P with h holes and R reflex vertices. Then:*

(i) *For any given scan direction, it is possible to solve the MUMC problem (for this scan direction) in time $O(n \log n + h \log^3 h)$.*
(ii) *It is possible to solve the general MUMC problem (that is, over all possible scan directions) in time $O(K \cdot (n \log n + h \log^3 h))$ time, where K is the number of edges in P's visibility graph.*

In additional $O(n + R^2)$ time, it is possible to output the complete monotone subdivision. From a representation of size $O(n)$, it is possible to output the boundary of the any desired component in $O(n+R)$ time.

6. Final path fairing

While the results of the previous sections provide a minimum number of components, the price we pay is that the number of Steiner points may be quite large. Although there exist worst-case inputs in which any optimal solution involves a quadratic number of Steiner points (for example, a narrow zig–zagging polygon with many merge SRVs on one side and many split SRVs on the other), such cases are unlikely to arise in typical applications. This raises the question of whether it is possible reduce the total number of Steiner points, while retaining the property of having the minimum number of monotone components.

Let S denote the subdivision induced by applying our flow-based solution and Lemma 4, but before adding in the PEPs. We consider a simple approach to reduce the total number of Steiner points along the FEPs of S, an operation we call *path fairing*. We do this by reducing the number of edges (or links) in the FEPs of S.

Our approach is based on a local refinement. Let D be the dual of S (as defined in Section 3). (In Fig. 24(a), D is the blue graph embedded in the subdivision S.) Let u and v be two adjacent nodes of D connected by an edge $e = (u, v)$ in D. These two nodes correspond to two components (that is, two adjacent subpolygons of P) that share a common boundary in the subdivision.

Starting at an arbitrary node of D, we can traverse the edges of D in any order, e.g., in depth-first order. For each edge $e = (u, v)$ in D, we will replace the common boundary of the two components corresponding to u and v with a minimum link x-monotone path between them. To do so, let P_e denote the union of the two polygons incident to u and v. Let ℓ and r denote the leftmost and rightmost endpoints of the common boundary of the two polygons, and let U_e be a maximal x-monotone polygon contained within P_e. For example, U_e can be defined to be the portion of P_e lying between two vertical lines, one through ℓ and one through r (see Fig. 24(a), the left polygon is U_e of the two subpolygons (of P) corresponding to the thick blue edge of D).

Suri [18] gave an algorithm for computing a minimum link path (not necessarily x-monotone) in any simple polygon without holes (U_e in our case). His algorithm runs in linear time, assuming that the polygon has been triangulated. We can produce such a triangulation in time linear in the complexity of U_e by a simple traversal of the boundary of U_e through the pseudo-trapezoidal map of P, which has already been computed. The next lemma states that any minimum link path generated by Suri's algorithm can be converted into a path that is x-monotone, without increasing the number of links. The proof can be found in [19].

Lemma 21. *Given an x-monotone polygon and two points ℓ and r within this polygon, there exists a minimum link path between ℓ and r that is x-monotone.*

Clearly, the running time of this path-fairing algorithm is proportional to the size of the final subdivision. As analyzed in Section 4, the size of the final subdivision is $O(n + rR)$, and is therefore the running time of the path fairing algorithm.

Since each application of this algorithm provides only a local improvement, it would be possible to repeat the algorithm a

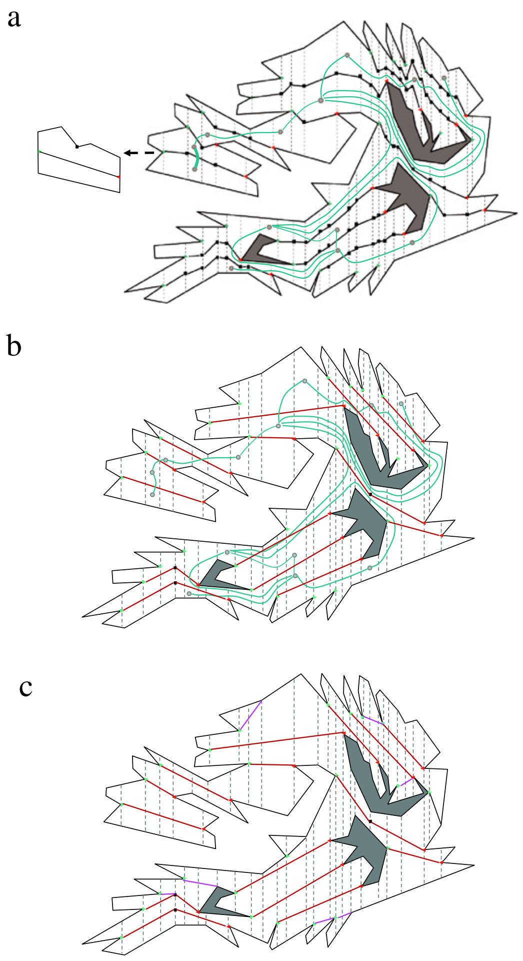


Fig. 24. Final path fairing.

constant number of times, or until no further reduction occurs in the number of Steiner points.

The path-fairing algorithm is illustrated in Fig. 24. D is the blue graph embedded in the final subdivision S. For the union of two polygons incident to the thick edge of D, we consider only the part between the walls incident to the two endpoints of the common boundary, as shown on the left the common boundary of the two polygons is replaced by a minimum link x-monotone path, a single link in fact.

After implementing this path fairing operation for every two adjacent polygons for one time, in Fig. 24(b), the fairing paths marked by the red links in S are shown. Finally, we need to add the PEPS to eliminate all the merge and split SRVs that are not the endpoints of any FEPs. This can be done by implementing a plane sweep paradigm on each polygon of the resulting S. Luckily, we have the pseudo-trapezoidal map already, therefore the PEPs can be added in linear time with respective to the size of the resulting S. The final subdivision is shown in Fig. 24(c), PEPs are marked by the pink links, for simplicity, graph D is removed.

7. Concluding remarks

We have resolved a longstanding open problem in the area of polygon decomposition, called MUMC, which involves computing the minimum number of uniformly monotone components (allowing arbitrary Steiner points). In contrast to most existing approaches to polygon decomposition, which have been based on dynamic programming, we show that the problem can be reduced to computing the maximum flow in a planar network with multiple sources and multiple sinks. Our approach is based on a novel collapsing process, which reduces the size of a planar network to one in which the total size of the network is proportional to the number of faces in the network.

Although the final subdivision is optimal with respect to the number of components, because of the Steiner points, it may have size that is superlinear in the input size. Our algorithm outputs a concise representation of the final decomposition, whose size is only linear in the size of the input polygon. From this representation, it is possible to extract any or all of the components of the decomposition in time proportional to the output size. This data structure may find applications in other polygon decomposition problems.

We have also presented a simple heuristic for path fairing, which reduces the number of Steiner points in the final decomposition. While this heuristic can significantly reduce the number of Steiner points, it does not necessarily achieve a global minimum. In particular, it does not alter the general structure (that is, the homotopy group) of the solution. This suggests a number of problems for future research. For example, among all solutions to the MUMC problem that achieve the minimum number components, what is the computational complexity of finding the subdivision that minimizes the number of Steiner points?

Acknowledgments

The second author's part of this work was supported by GRF grant 614309. The third author's work has been supported by the National Science Foundation under grant CCF-1117259 and the Office of Naval Research under grant N00014-08-1-1015.

References

[1] Preparata FP, Shamos MI. Computational geometry: an introduction. Berlin: Springer-Verlag; 1990.
[2] Chazelle B. Triangulating a simple polygon in linear time. Discrete & Computational Geometry 1991;6:485–524.
[3] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational geometry: algorithms and applications. 3rd ed. Berlin, Germany: Springer-Verlag; 2008.
[4] Chazelle B, Dobkin DP. Optimal convex decompositions. In: Toussaint GT, editor. Computational geometry. Amsterdam, Netherlands: North-Holland; 1985. p. 63–133.
[5] Keil JM. Polygon decomposition. In: Handbook of computational geometry. North-Holland Publishing Co.; 2000. p. 491–518.
[6] Dwivedi R, Kovacevic R. Automated torch path planning using polygon subdivision for solid freeform fabrication based on welding. Journal of Manufacturing Systems 2004;23:278–91.
[7] Nahar S, Sahni S. Fast algorithm for polygon decomposition. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 1988;7:473–83.
[8] Lee DT, Preparata FP. Location of a point in a planar subdivision and its applications. SIAM Journal on Computing 1977;6:594–606.
[9] Garey MR, Johnson DS, Preparata FP, Tarjan RE. Triangulating a simple polygon. Information Processing Letters 1978;7:175–9.
[10] Keil JM. Decomposing a polygon into simpler components. SIAM Journal on Computing 1985;14:799–817.
[11] Liu R, Ntafos S. On decomposing polygons into uniformly monotone components. Information Processing Letters 1988;27:85–9.
[12] Kleinberg J, Tardos E. Algorithm design. Boston, MA: Addison-Wesley; 2006.
[13] Preparata FP, Supowit KJ. Testing a polygon for monotonicity. Information Processing Letters 1981;12:161–4.
[14] Arkin EM, Connelly R, Mitchell JSB. On monotone paths among obstacles with applications to planning assemblies. In: Proceedings of the 5th annual symposium on computational geometry. 1989. p. 334–43.
[15] Borradaile G, Klein P, Mozes S, Nussbaum Y, Wulff-Nilsen C. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In: Proceedings of the 52nd annual IEEE symposium on foundations of computer science. 2011. p. 170–79.
[16] Ahuja RK, Magnanti TL, Orlin JB. Network flows: theory, algorithms, and applications. Englewood Cliffs, NJ: Prentice-Hall; 1993.
[17] van Lint JH, Wilson RM. A course in combinatorics. Cambridge; 1992.
[18] Suri S. A linear time algorithm with minimum link paths inside a simple polygon. Computer Vision, Graphics, and Image Processing 1986;35:99–110.
[19] Wei X, Joneja A. On minimum link monotone path problems. Transactions of the ASME Journal of Computing and Information Science in Engineering 2011; 11(3):031002. http://dx.doi.org/10.1115/1.3615687.