

# The Parametric Closure Problem

David Eppstein

Computer Science Department, Univ. of California, Irvine, USA

**Abstract.** We define the *parametric closure problem*, in which the input is a partially ordered set whose elements have linearly varying weights and the goal is to compute the sequence of minimum-weight downsets of the partial order as the weights vary. We give polynomial time solutions to many important special cases of this problem including semiorders, reachability orders of bounded-treewidth graphs, partial orders of bounded width, and series-parallel partial orders. Our result for series-parallel orders provides a significant generalization of a previous result of Carlson and Eppstein on bicriterion subtree problems.

## 1 Introduction

*Parametric optimization* problems are a variation on classical combinatorial optimization problems such as shortest paths or minimum spanning trees, in which the input weights are not fixed numbers, but vary as functions of a parameter. Different parameter settings will give different weights and different optimal solutions; the goal is to list these solutions and the intervals of parameter values within which they are optimal. As a simple example, consider maintaining the minimum of  $n$  input values, which change as a parameter controlling these values changes. This parametric minimum problem can be formalized as the problem of constructing the *lower envelope* of functions that map the parameter value to each input value. When these are linear functions this is the problem of constructing the lower envelope of lines, equivalent by projective duality to a planar convex hull [31]; the lower envelope has linear complexity and can be constructed in time  $O(n \log n)$ .

As well as the obvious applications of parametric optimization to real-world problems with time-varying but predictable data (such as rush-hour route planning), parametric optimization problems have another class of applications, to *bicriterion optimization*. In bicriterion problems, each input element has two numbers associated with it. A solution value is obtained by summing the first number for each selected element, separately summing the second number for each selected element, and evaluating a nonlinear combination of these two sums. For instance, each element's two numbers might be interpreted as the  $x$  and  $y$  coordinates of a point in the plane associated with the element, and we might wish to find the solution whose summed  $x$  and  $y$  coordinates give a point as close to the origin as possible. The two numbers might represent the mean and variance of a normal distribution, and we might wish to optimize some function of the summed distribution. The two numbers on each element might represent an initial investment cost and expected profit of a business opportunity, and we might wish to find a feasible combination of opportunities that maximizes the return on investment. Or, the two numbers might represent the cost and log-likelihood of failure of a communications link, and we might wish to find a low-cost communications network with high probability of success. Many natural bicriterion optimization problems of this type can be expressed as finding the maximum of a *quasiconvex function* of the two sums (a function whose lower level sets are convex sets) or equivalently as finding the minimum of a quasiconcave function of the two sums. When this is the case, the optimal solution can always be obtained as one of the solutions of a parametric problem on only one variable, defined by re-interpreting the two numbers associated with each input element as the slope and  $y$ -intercept of a linear function that gives the weight of that element (a single number) as a function of the parameter [9, 25]. In this way, any algorithm for

solving a parametric optimization problem can also be used to solve bicriterion versions of the same type of optimization problem. Even though the bicriterion problem might combine its two numbers in a nonlinear way, the corresponding parametric problem uses linearly varying edge weights.

In this paper we formulate and provide the first study of the *parametric closure problem*, the natural parametric variant of a classical optimization problem, the *maximum closure problem* [21,35].

## 1.1 Closures and parametric closures

A closure in a directed graph is a subset of vertices such that all edges from a vertex in the subset go to another vertex in the subset; the maximum closure problem is the problem of finding the highest-weight closure in a vertex-weighted graph. Equivalently we seek the highest-weight *downset* of a weighted partial order, where a downset is a subset of the elements of a partial order such that if  $x < y$  in the order, and  $y$  belongs to the subset, then  $x$  also belongs to the subset. A partial order can be converted to an equivalent directed graph by considering each element of the order as a vertex, with an edge from  $y$  to  $x$  whenever  $x < y$  in the order (note the reversed edge direction from the more usual conventions). In the other direction, given a directed graph, one may obtain an equivalent partial order on the strongly connected components of the graph, where component  $x$  is less than component  $y$  in the partial order whenever there is a path from a vertex in  $y$  to a vertex in  $x$  in the graph.

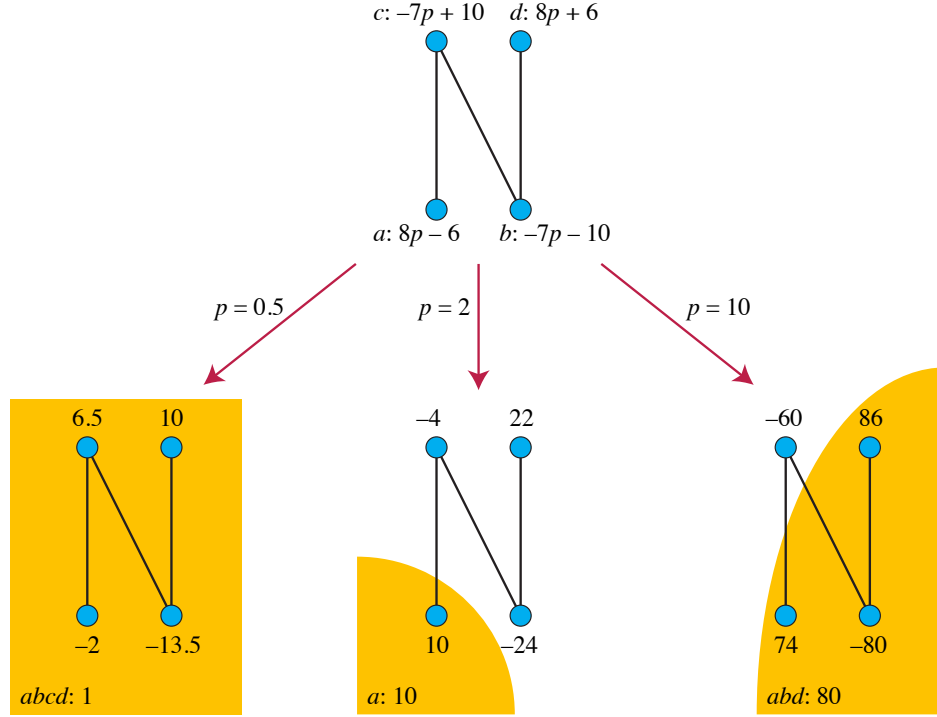
One of the classical applications of this problem involves open pit mining [27], where the vertices of a directed graph represent blocks of ore or of covering material that must be removed to reach the ore, edges represent ordering constraints on the removal of these blocks, and the weight of each vertex represents the net profit or loss to be made by removing that block. Other applications of this problem include military attack planning [34], freight depot placement [4, 37], scheduling with precedence constraints [11, 26], image segmentation [2, 19], stable marriage with maximum satisfaction [22], and treemap construction in information visualization [8]. Maximum closures can be found in polynomial time by a reduction to maximum flow [20, 35] or by direct algorithms [16].

In the parametric closure problem, we assign weights to the vertices of a directed graph (or the elements of a partial order) that vary linearly as functions of a parameter, and we seek the closures (or downsets) that have maximum weight for each possible value of the parameter (Figure 1). For instance, in the open pit mining problem, the profit or loss of a block of ore will likely vary as a function of the current price of the refined commodity produced from the ore. So, a parametric version of the open pit mining problem can determine a range of optimal mining strategies, depending on how future commodity prices vary. As described above, an algorithm for parametric closures can also solve bicriterion closure problems of maximizing a quasiconvex function (or minimizing a quasiconcave function) of two sums of values. Although we have not been able to resolve the complexity of the parametric closure problem in the general case, we prove near-linear or polynomial complexity for several important special cases of this problem.

## 1.2 Related work

Previous work on parametric optimization has considered parametric versions of two standard network optimization problems, the minimum spanning tree problem and the shortest path problem. The parametric minimum spanning tree problem (with linear edge weights) has polynomially many solutions that can be constructed in polynomial time [1, 15, 18]. In contrast, the parametric shortest path problem is not polynomial, at least if the output must be represented as an explicit list of paths: it has a number of solutions and running time that are exponential in  $\log^2 n$  on  $n$ -vertex graphs [10].

We do not know of previous work on the general parametric closure problem, but two previous papers can be seen in retrospect as solving special cases:

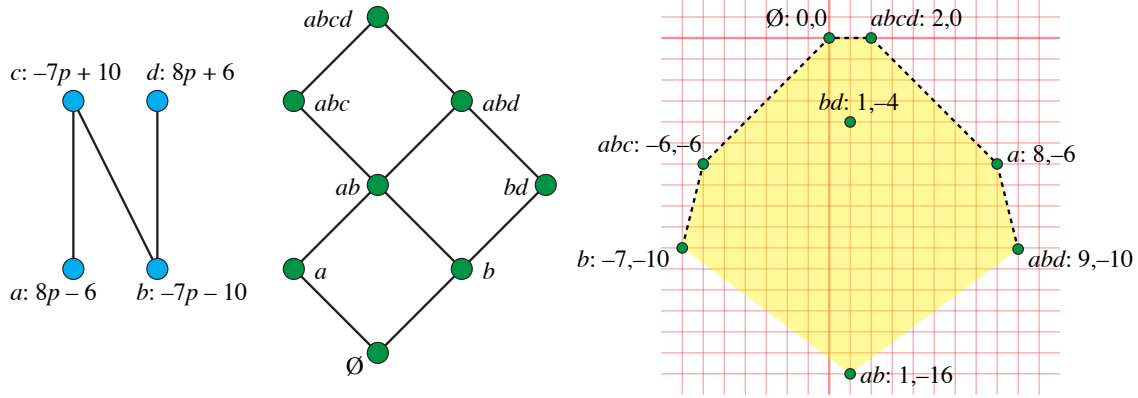


**Fig. 1.** In a partially ordered set with weights that vary linearly as a function of the parameter  $p$ , different choices of  $p$  lead to different maximum-weight downsets.

- Lawler [26] studied scheduling to minimize weighted completion time with precedence constraints. He sought the closure that maximizes the ratio  $x/y$  of the priority  $x$  and processing time  $y$  of a job or set of jobs, and used this closure to decompose instances of this problem into smaller subproblems. Instead of using the reduction from bicriterion to parametric problems, Lawler showed that the optimal closure can be found in polynomial time by a binary search where each step involves the solution of a weighted closure problem. Replacing the binary search by Megiddo’s parametric search [32] would make this algorithm strongly polynomial. Both search methods depend on the specific properties of the ratio function, however, and cannot be extended to other bicriterion problems.
- Carlson and Eppstein [9] consider bicriterion versions of the problem of finding the best subtree (containing the root) of a given rooted tree with weighted edges. As they show, many such problems can be solved in time  $O(n \log n)$ . Although Carlson and Eppstein did not formulate their problem as a closure problem, the root-containing subtrees can be seen as closures for a directed version of the tree in which each edge is directed towards the root. The reachability ordering on this directed tree is an example of a series-parallel partial order, and we greatly generalize the results of Carlson and Eppstein in our new results on parametric closures for arbitrary series-parallel partial orders.

### 1.3 Parametric optimization as an implicit convex hull problem

Parametric optimization problems can be formulated dually, as problems of computing convex hulls of implicitly defined two-dimensional point sets (Figure 2). Suppose we are given a parametric optimization problem in which weight of element  $i$  is a linear function  $a_i\lambda + b_i$  of a parameter  $\lambda$ ,



**Fig. 2.** An instance of the parametric closure problem. Left: The Hasse diagram of a partially ordered set  $N$  of four elements, each with a weight that varies linearly with a parameter  $p$ . Center: The distributive lattice  $\text{down}(N)$  of downsets of  $N$ . Right: The point set  $\text{project}(\text{down}(N))$  and its convex hull. The upper hull (dashed) gives in left-to-right order the sequence of six distinct maximum-weight closures as the parameter  $p$  varies continuously from  $-\infty$  to  $+\infty$ .

and in which the weight of a candidate solution  $S$  (a subset of elements, constrained by the specific optimization problem in question) is the sum of these functions. Then the solution value is also a linear function, whose coefficients are the sums of the element coefficients:

$$\sum_{i \in S} a_i \lambda + b_i = \left( \sum_{i \in S} a_i \right) \lambda + \left( \sum_{i \in S} b_i \right).$$

Instead of interpreting the numbers  $a_i$  and  $b_i$  as coefficients of linear functions, we may re-interpret the same two numbers as the  $x$  and  $y$  coordinates (respectively) of points in the Euclidean plane. In this way any family  $\mathcal{F}$  of candidate solutions determines a planar point set, in which each set in  $\mathcal{F}$  corresponds to the point given by the sum of its elements' coefficients. We call this point set  $\text{project}(\mathcal{F})$ , because the sets in  $\mathcal{F}$  can be thought of as vertices of a hypercube  $Q_n = \{0, 1\}^n$  whose dimension is the number of input elements, and  $\text{project}$  determines a linear projection from these vertices to the Euclidean plane.

Let  $\text{hull}(\text{project}(\mathcal{F}))$  denote the convex hull of this projected planar point set. Then for each parameter value the set in  $\mathcal{F}$  minimizing or maximizing the parameterized weight corresponds by projective duality to a vertex of the hull, and the same is true for the maximizer of any quasiconvex function of the two sums of coefficients  $a_i$  and  $b_i$ . Thus, parametric optimization can be reformulated as the problem of constructing this convex hull, and bicriterion optimization can be solved by choosing the best hull vertex.

#### 1.4 New results

For an arbitrary partially ordered set  $P$ , define  $\text{down}(P)$  to be the family of downsets of  $P$ . Let  $P$  be parametrically weighted, so that  $\text{project}(\text{down}(P))$  is defined. As a convenient abbreviation, we define  $\text{polygon}(P) = \text{hull}(\text{project}(\text{down}(P)))$ . This convex polygon represents the solution to the parametric closure problem for the given weights on  $P$ .

We consider the following classes of partially ordered sets. For each partial order  $P$  in one of these classes, we prove polynomial bounds on the complexity of  $\text{polygon}(P)$  and on the time for constructing  $\text{polygon}(P)$ . These results imply the same time bounds for parametric optimization over  $P$  and for maximizing a quasiconvex function (bicriterion optimization) over  $P$ .

**Semiorders.** This class of partial orders was introduced to model human preferences [30] in which each element can be associated with a numerical value, pairs of elements whose values are within a fixed margin of error are incomparable, and farther-apart pairs are ordered by their numerical values. They are equivalently the interval orders with intervals of unit length, or the proper interval orders (interval orders in which no interval contains another). For such orderings, we give in Section 3 a bound of  $O(n \log n)$  on the complexity of  $\text{polygon}(P)$  and we show that it can be constructed in time  $O(n \log^2 n)$  using an algorithm based on the quadtree data structure.

**Series-parallel partial orders.** These are orders formed recursively from smaller orders of the same type by two operations: series compositions (in which all elements from one order are placed earlier in the combined ordering than all elements of the other order) and parallel compositions (in which pairs of one element from each ordering are incomparable). These orderings have been applied for instance in scheduling applications by Lawler [26]. For such orderings, the sets of the form  $\text{polygon}(P)$  have a corresponding recursive construction by two operations: the convex hull of the union of two convex polygons, and the Minkowski sum of two convex polygons. It follows that  $\text{polygon}(P)$  has complexity  $O(n)$ . This construction does not immediately lead to a fast construction algorithm, but in Section 4 we adapt a splay tree data structure [39] to construct  $\text{polygon}(P)$  in time  $O(n \log n)$ . Our previous results for optimal subtrees [9] follow as a special case of this result.

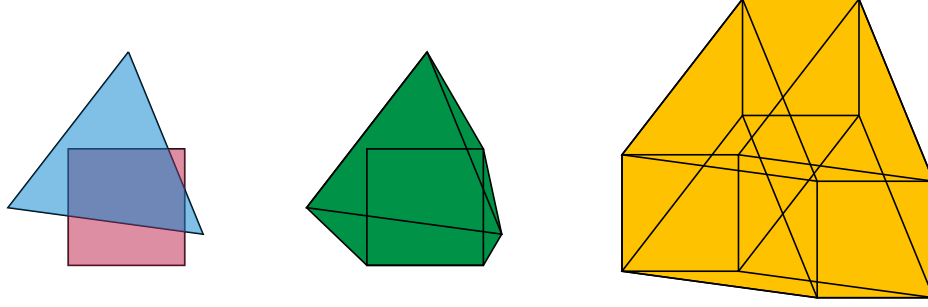
**Bounded treewidth.** Suppose that partial order  $P$  has  $n$  elements and its transitive reduction (the *covering graph* of  $P$ ) forms a directed acyclic graph whose undirected version has treewidth  $t$ . (For prior work on treewidth of partial orders, see [24]; for prior work on parametric optimization on graphs of bounded treewidth, see [17].) Then we show in Section 5 that  $\text{polygon}(P)$  has polynomially many vertices, with exponent  $t + 2$ , and that it can be constructed in polynomial time.

**Incidence posets.** The incidence poset of a graph  $G$  has the vertices and edges as elements, with an order relation  $x \leq y$  whenever  $x$  is an endpoint of  $y$ . One of the initial applications for the closure problem concerned the design of freight delivery systems in which a certain profit could be expected from each of a set of point-to-point routes in the system, but at the cost of setting up depots at each endpoint of the routes [4,37]; this can be modeled with an incidence poset for a graph with a vertex at each depot location and an edge for each potential route. Since the profits and costs have different timeframes, it is reasonable to combine them in a nonlinear way, giving a bicriterion closure problem. The transitive reduction of an incidence poset is a subdivision of  $G$  with the same treewidth, so our technique for partial orders of bounded treewidth also applies to incidence posets of graphs of bounded treewidth.

**Fences, generalized fences, and polytrees.** The fences or zigzag posets are partial orders whose transitive reduction is a path with alternating edge orientations. A *generalized fence* may be either an oriented path (an up-down poset) [33] or an oriented tree (polytree) [38]. Polynomial bounds on the complexity and construction time for  $\text{polygon}(P)$  for all of these classes of partial orders follow from the result for treewidth  $t = 1$ . However, in these cases we simplify the bounded-treewidth construction and provide tighter bounds on these quantities (Theorem 3 and Theorem 4).

**Bounded width.** The *width* of a partial order is the maximum number of elements in an *antichain*, a set of mutually-incomparable elements. Low-width partial orders arise, for instance, in the edit histories of version control repositories [5]. The treewidth of a partial order is less than twice its width,<sup>1</sup> but partial orders of width  $w$  have  $O(n^w)$  downsets, tighter than the bound that would be obtained by using treewidth. We use quadtrees to show more strongly in Section 6 that in

<sup>1</sup> To obtain a path-decomposition of width  $2w - 1$  from a partial order of width  $w$ , consider any linear extension of the partial order, partition the partial order into a downset and an upset at each position of the linear extension,



**Fig. 3.** Two convex polygons  $P$  and  $Q$  (left), the convex hull of their union  $P \uplus Q$  (center), and their Minkowski sum  $P \oplus Q$  (right).

this case  $\text{polygon}(P)$  has  $O(n^{w-1} + n \log n)$  vertices and can be constructed in time within a logarithmic factor of this bound.

We have been unable to obtain an example of a family of partial orders with a nonlinear lower bound on the complexity of  $\text{polygon}(P)$ , nor have we been able to obtain a nontrivial upper bound on the hull complexity for unrestricted partial orders. Additionally, we have been unable to obtain polynomial bounds on the hull complexity of the above types of partial orders for more than one parameter (i.e., for weights of dimension higher than two). We also do not know of any computational complexity bounds (such as NP-hardness) for the parametric closure problem for any class of partial orders in any finite dimension. We leave these problems open for future research.

## 2 Preliminaries: Minkowski sums and hulls of unions

Our results on the complexity of the convex polygons  $\text{polygon}(P)$  associated with a partial order hinge on decomposing these polygons recursively into combinations of simpler polygons. To do this, we use two natural geometric operations that combine pairs of convex polygons to produce more complex convex polygons.

**Definition 1.** We define a convex polygon to be the convex hull of a nonempty finite set of points in the Euclidean plane. A vertex of the polygon is a point that can be obtained as the intersection of the polygon with a closed halfplane, and an edge of the polygon is a line segment that can be obtained as the intersection of the polygon with a closed halfplane. This definition requires us to include two degenerate special cases: we consider a single point to be a degenerate convex polygon with one vertex and no edges, and we consider a line segment to be a degenerate convex polygon with two vertices and one edge.

**Definition 2.** For any two convex polygons  $P$  and  $Q$ , let  $P \oplus Q$  denote the Minkowski sum of  $P$  and  $Q$  (the set of points that are the vector sum of a point in  $P$  and a point in  $Q$ ), and let  $P \uplus Q$  denote the convex hull of the union of  $P$  and  $Q$  (Figure 3).

**Lemma 1 (folklore).** If convex polygons  $P$  and  $Q$  have  $p$  and  $q$  vertices respectively, then  $P \oplus Q$  and  $P \uplus Q$  have at most  $p + q$  vertices, and can be constructed from  $P$  and  $Q$  in time  $O(p + q)$ .

---

and form the sequence of sets that are the union of the maxima of a downset and the minima of the complementary upset.

*Proof.* The complexity bound on  $P \oplus Q$  follows from the fact that the set of edge orientations of  $P \oplus Q$  is the union of the sets of edge orientations of  $P$  and of  $Q$ . Therefore,  $P \oplus Q$  has at most as many edges as the sum of the numbers of edges in  $P$  and  $Q$ . The result follows from the fact that in any convex polygon, the numbers of vertices and edges are equal (except for the degenerate special cases, with one more vertex than edges). To compute  $P \oplus Q$  from  $P$  and  $Q$ , we may merge the two cyclically-ordered lists of edge orientations of the two polygons in linear time, and then use the sorted list to trace out the boundary of the combined polygon.

Similarly, the complexity bound on  $P \uplus Q$  follows from the fact that its vertex set is a subset of the union of the vertices in  $P$  and the vertices in  $Q$ . Therefore, its number of vertices is at most the sum of the numbers of vertices of  $P$  and  $Q$ . To compute  $P \uplus Q$  from  $P$  and  $Q$  in linear time, it is convenient to partition each convex hull into its *lower hull* and *upper hull*, two monotone polygonal chains, by splitting it at the leftmost and rightmost vertex of the hull. We may sort the vertices of the upper hulls by doing a single merge in linear time, apply Graham scan to the sorted list, and do the same for the lower hulls.  $\square$

**Corollary 1.** *Suppose that  $P$  is a convex polygon, described as a formula that combines a set of  $n$  points in the plane into a single polygon using a sequence of  $\oplus$  and  $\uplus$  operations. Suppose in addition that, when written as an expression tree, this formula has height  $h$ . Then  $P$  has at most  $n$  vertices and it may be constructed from the formula in time  $O(nh)$ .*

More complex data structures can reduce this time to  $O(n \log n)$ ; see Section 4.

In higher dimensions, the convex hull of  $n$  points and Minkowski sum of  $n$  line segments both have polynomial complexity with an exponent that depends linearly on the dimension. However, we do not know of an analogous bound on the complexity of convex sets formed by mixing Minkowski sum and hull-of-union operations. If such a bound held, we could extend our results on parametric closures to the corresponding higher dimensional problems.

### 3 Semiorders

A *semiorder* is a type of partial order defined by Luce [30] to model human preferences. Each element of the order can be associated with a numerical value, which in the application to preference modeling is the *utility* of that element to the person whose preferences are being modeled. For pairs of items whose utilities are sufficiently far from each other, the ordering of the two items in the semiorder is the same as the numerical ordering of their utilities. However, items whose utilities are within some (global) margin of error of each other are incomparable in the semiorder. More formally:

**Definition 3.** *Let a collection of items  $x_i$  be given, with numerical utilities  $u_i$ , together with a (global) threshold  $\theta > 0$ . Then this information determines a partial order in which  $x_i \leq x_j$  whenever  $u_i \leq u_j - \theta$ . The partial orders that can be obtained in this way are called the semiorders. We will call  $\theta$  the margin of error of the semiorder.*

(We note that some authors use irreflexive binary relations, instead of partial orders, to define semiorders; this distinction will not be important for us.)

Similar concepts of comparisons of numerical values with margins of error give rise to semiorders in many other areas of science and statistics [36]. For efficient computations on semiorders we will assume that the utility values of each element are part of the input to an algorithm, and that the margin of error has been normalized to one. For instance, the semiorder  $N$  of Figure 2 can be represented as a semiorder with utilities  $2/3$ ,  $0$ ,  $2$ , and  $4/3$  for  $a$ ,  $b$ ,  $c$ , and  $d$  respectively. With this information in hand, the comparison between any two elements can be determined in constant

time. If only the ordering itself is given, numerical utility values can be constructed from it in time  $O(n^2)$  [3].

The concept of a downset is particularly natural for a semiorder: it is a set of elements whose utility values could lie below a sharp numerical threshold, after perturbing each utility value by at most half the margin of error. In this way, the closure problem (the problem of finding a maximum weight downset) can alternatively be interpreted as the problem of finding the maximum possible discrepancy of a one-dimensional weighted point set in which the location of each point is known imprecisely. Thus, this problem is related to several other recent works on geometric computations with imprecise points (see, e.g., [28, 29]). Semiorders may have exponentially many downsets; for instance, if all items have utilities that are within one unit of each other, all sets are downsets. Nevertheless, as we show in this section, if  $S$  is a semiorder, then the complexity of  $\text{polygon}(S)$  (the number of downsets that are optimal for some parameter value) is  $O(n \log n)$ .

### 3.1 Mapping downsets to a grid

If  $S$  is any parametrically weighted semiorder, we may write the sorted order of the utility values of elements of  $S$  as  $u_0, u_1, \dots, u_{n-1}$  where  $n = |S|$ , and we may write the elements themselves (in the same order) as  $x_0, x_1, \dots, x_{n-1}$ . By scaling the utility values we may assume without loss of generality that the threshold  $\theta$  used to define a semiorder from these values is set as  $\theta = 1$ . By padding  $S$  with items that have a fixed zero weight and a utility that is smaller than that of the elements by more than the margin of error, we may additionally assume without loss of generality that  $n$  is a power of two without changing the values of the parametric closure problem on  $S$ .

It is convenient to parameterize downsets by pairs of integers, as follows.

**Definition 4.** Let  $L$  be an arbitrary downset in  $\text{down}(S)$ . Let  $j$  be the largest index of an element  $x_j$  of  $L$ . Let  $i$  be the smallest index of an element  $x_i$  such that  $x_i$  does not belong to  $L$  and  $i < j$ , or  $-1$  if no such element exists. Define  $\text{extremes}(L)$  to be the pair of integers  $(i + 1, j)$ .

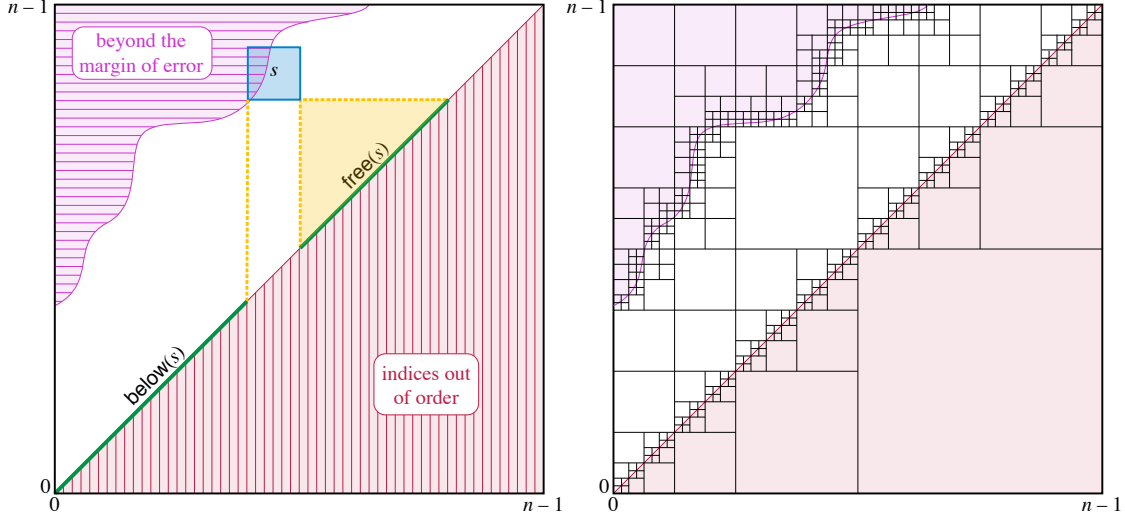
Thus,  $\text{extremes}$  maps the family  $\text{down}(S)$  to the integer grid  $[0, n - 1]^2$ . The mapping is many-to-one: potentially, many downsets may be mapped to each grid point. However, not every grid point is in the image of  $\text{down}(S)$ . In particular, a point  $(i, j)$  with  $i > j$  cannot be the image of a downset, because the element defining the first coordinate of  $\text{extremes}$  must have an index smaller than the element defining the second coordinate. Additionally, when  $i > 0$ , a point  $(i, j)$  with  $u_{i-1} < u_j - 1$  (i.e. with utility values that are beyond the margin of error for the semiorder) also cannot be the image of a downset, because in this case  $x_{i-1} \leq x_j$  in the semiorder, so every downset that includes  $x_j$  also includes  $x_{i-1}$ . Thus, the image of  $\text{extremes}$  lies in an orthogonally convex subset of the grid, bounded below by its main diagonal and above by a monotone curve (Figure 4).

**Definition 5.** Let  $s$  be any square subset of the integer grid  $[0, n - 1]^2$ , and define  $\text{subproblem}(s)$  to be the partially-ordered subset of the semiorder  $S$  consisting of the elements whose indices are among the rows and columns of  $s$ . Define  $\text{free}(s)$  to be the (unordered) set of elements of  $S$  that do not belong to  $\text{subproblem}(s)$ , but whose indices are between pairs of indices that belong to  $\text{subproblem}(s)$ . Define  $\text{below}(s)$  to be the set of elements of  $S$  whose indices are smaller than all elements of  $\text{subproblem}(s)$ . (See Figure 4, left, for an example.)

### 3.2 Decomposition of grid squares

These definitions allow us to decompose the downsets that are mapped by  $\text{extremes}$  to the given square  $s$ .





**Fig. 4.** The grid  $[0, n-1]^2$ , with the two regions that cannot be part of the image of **extremes**. The left image shows a square subproblem  $s$  and  $\text{free}(s)$ ; the right image shows the quadtree decomposition of the grid used to prove Theorem 1.

**Lemma 2.** *Given a square  $s$ , suppose that the subfamily  $\mathcal{F}$  of  $\text{down}(S)$  that is mapped by **extremes** to  $s$  is nonempty. Then each set in  $\mathcal{F}$  is the disjoint union of three sets: a nonempty downset of  $\text{subproblem}(s)$ , the set  $\text{below}(s)$ , and an arbitrary subset of  $\text{free}(s)$ .*

*Proof.* Any downset of  $S$  must remain a downset in any subset of  $S$ , and in particular its intersection with  $\text{subproblem}(s)$  must also be a downset. Additionally, it is not possible for a set in  $\mathcal{F}$  to omit any member of  $\text{below}(s)$ , nor to include any element outside  $\text{subproblem}(s) \cup \text{below}(s) \cup \text{free}(s)$ , for then it would not be mapped into  $s$  by **extremes**. Similarly, the condition on the row index of the largest element of  $\text{subproblem}(s)$  must be met, or again **extremes** would map the given set outside of  $s$ . Therefore, every set in  $\mathcal{F}$  has the form described.  $\square$

Conversely, let a set be formed as the disjoint union of a nonempty downset of  $\text{subproblem}(s)$ ,  $\text{below}(s)$ , and an arbitrary subset of  $\text{free}(s)$ . Then this set is necessarily a downset, although it might not be mapped by **extremes** to  $s$  (depending on the choice of the downset of  $\text{subproblem}(s)$  in the disjoint union).

We can decompose the convex hull of the projections of these downsets into a contribution from the subproblem of  $s$  and another contribution from the free set of  $s$ . The contribution from the free set has a simple structure based on Minkowski sums:

**Lemma 3.** *For any square  $s$ , let  $\text{powerset}(\text{free}(s))$  be the family of all subsets of  $\text{free}(s)$ . Let weight function  $w : S \rightarrow \mathbb{R}^2$  define a projection **project** from families of sets to point sets in  $\mathbb{R}^2$ . Then  $\text{project}(\text{powerset}(\text{free}(s)))$  is the Minkowski sum of the sets  $\{(0, 0), w(x_i)\}$  for  $x_i \in \text{free}(s)$ . Its convex hull is a centrally symmetric convex polygon  $\text{hull}(\text{project}(\text{powerset}(\text{free}(s))))$  (the Minkowski sum of the corresponding line segments) with at most  $k = 2|\text{free}(s)|$  sides (fewer if some of these line segments are collinear), and can be constructed in time  $O(k \log k)$ .*

*Proof.* The fact that  $\text{project}(\text{powerset}(\text{free}(s)))$  is a Minkowski sum of line segments can be proved by induction on the number of members of  $\text{free}(s)$ : for any particular element  $x$ , the subfamilies of downsets that include or exclude  $x$  project to translates of the same Minkowski sum, by the induction hypothesis, and the convex hull of the union of these two translates is the same thing as

the Minkowski sum with one more line segment. The bounds on the number of sides and construction time follow immediately from Corollary 1, using the fact that the Minkowski sum of these segments can be expressed as a tree of binary Minkowski sums of logarithmic height.  $\square$

The  $O(k \log k)$  time bound in Lemma 3 can be reduced. The algorithm described, using a balanced tree of binary Minkowski sums, can be interpreted as performing a merge sort on the slopes of the line segments whose Minkowski sum is the desired hull by their slopes. If these segments were already sorted, the Minkowski sum could be constructed in linear time. And, in this case, it is possible to preprocess the input so that, for any contiguous subsequence of the elements, we can sort the corresponding line segments by their slopes quickly using integer sorting algorithms. However, we omit this speedup, as it adds complication to our overall algorithm without improving its total time.

The same proof used for Lemma 3 also allows us to quickly compute  $\text{polygon}(\text{subproblem}(s))$  for any square  $s$  that is entirely within the margin of error, using the fact that the family of downsets of this subproblem is just its power set.

**Lemma 4.** *Let  $s$  be a square of the grid defined by a given semiorder that is entirely within the margin of error of the semiorder, of side length  $\ell$ . Then  $\text{polygon}(\text{subproblem}(s))$  is a polygon with  $O(\ell)$  sides and can be computed in time  $O(\ell \log \ell)$ .*

Computing  $\text{polygon}(\text{subproblem}(s))$  for the remaining squares is trickier, but may be performed by decomposing the polygon into polygons of the same type for four smaller squares.

**Lemma 5.** *Let  $s$  be a square in the grid  $[0, n-1]^2$ , containing an even number of grid points, and subdivide  $s$  into four congruent smaller squares  $s_i$  ( $0 \leq i < 4$ ). Let  $\text{polygon}(\text{subproblem}(s))$  have  $c$  vertices, define  $c_i$  in the same way for each  $s_i$ , and let  $\ell$  be the side length of  $s$ . Then  $c \leq \sum c_i + O(\ell)$ , and  $\text{polygon}(\text{subproblem}(s))$  can be constructed from the corresponding hulls for the smaller squares in time  $O(\sum c_i + \ell \log \ell)$ .*

*Proof.* For each smaller square  $s_i$ , define  $H_i$  to be the polygon

$$\text{polygon}(\text{subproblem}(s_i)) \oplus \text{project}(\text{powerset}(\text{free}(s_i) \cap \text{subproblem}(s)))$$

translated by adding all of the weights of elements in  $\text{below}(s_i) \cap \text{subproblem}(s)$ .

Then each vertex of this polygon  $H_i$  represents a downset of  $\text{subproblem}(s)$ . By Lemma 2 (viewing  $s_i$  as a subproblem of  $\text{subproblem}(s)$ ) every downset of  $\text{subproblem}(s)$  that is mapped by  $\text{extremes}$  into  $s_i$  is included in  $H_i$ . (This polygon may also include some other downsets mapped into  $s$  but not  $s_i$ , but this is not problematic.) Therefore, we can construct  $\text{polygon}(\text{subproblem}(s))$  itself as

$$\text{polygon}(\text{subproblem}(s)) = H_0 \uplus H_1 \uplus H_2 \uplus H_3.$$

Thus, we have found a formula that expresses  $\text{polygon}(\text{subproblem}(s))$  using a constant number of Minkowski sums and unions of hulls of the corresponding hulls for smaller squares, together with free subproblems of total size  $O(\ell)$ . The result follows by using Lemma 3 to construct the polygons for the free subproblems and Corollary 1 to combine the resulting polygons into a single polygon.  $\square$

### 3.3 Semiorder algorithm

Putting it all together, we can use the observations above to decompose  $\text{polygon}(S)$  into a combination of similarly-computed polygons for smaller grid squares, recursively decomposed into even smaller squares. This leads to an algorithm that performs this decomposition and then uses it in a bottom-up order to construct the polygons of larger subproblems from smaller ones, eventually producing the solution to the whole problem.

**Theorem 1.** *If  $S$  is a semiorder with  $n$  elements  $x_i$ , specified with their utility values  $u_i$  and a system of two-dimensional weights  $w(x_i)$ , then  $\text{polygon}(S)$  has complexity  $O(n \log n)$  and can be constructed in time  $O(n \log^2 n)$ .*

*Proof.* We sort the utility values, pad  $n$  to the next larger power of two if necessary and form a quadtree decomposition of the grid  $[0, n-1]^2$  (as shown in Figure 4, right). For each square  $s$  of this quadtree, we associate a convex polygon (or empty set)  $\text{polygon}(\text{subproblem}(s))$  computed according to the following cases:

- If  $s$  is a subset of the grid points for which  $i > j$ , or for which  $u_{i-1} < u_j - 1$ , then no downsets are mapped into  $s$  by **extremes**. We associate square  $s$  with the empty set.
- If  $s$  is a subset of the grid points for which  $i \leq j$  and  $u_{i-1} \geq u_j - 1$ , then every two elements of  $\text{subproblem}(s)$  are incomparable. In this case, we associate square  $s$  with the polygon  $\text{hull}(\text{project}(\text{powerset}(\text{subproblem}(s))))$  computed according to Lemma 4.
- Otherwise, we split  $s$  into four smaller squares. We construct the polygon associated with  $s$  by using Lemma 5 to combine the polygons associated with its children.

It follows by induction that the total complexity of the polygon constructed at any square  $s$  of the quadtree is  $O(\sum \ell_i)$ , and the total time for constructing it is  $O(\sum \ell_i \log n)$ , where  $\ell_i$  is the side length of the  $i$ th square of the quadtree and the sum ranges over all descendants of  $s$ . Here, the  $O(\ell_i \log n)$  contribution to the time bound includes not only the terms of the form  $O(\ell_i \log \ell_i)$  appearing in Lemma 3 and Lemma 4, but also the amount of time spent combining the polygon for this square with other polygons at  $O(\log n)$  higher levels of the recursive subdivision.

As a base case for the induction, a square containing only a single grid point is associated with a subproblem with one element, with only one downset that maps to that grid point, and a degenerate convex polygon with a single vertex. The polygon constructed at the root of the quadtree is the desired output, and it follows that it has combinatorial complexity and time complexity of the same form, with a sum ranging over all quadtree squares.

The conditions  $i > j$  and  $u_{i-1} < u_j - 1$  define two monotone curves through the grid, and we split a quadtree square only when it is crossed by one of these two curves. It follows that the squares of side length  $\ell$  that are subdivided as part of this algorithm themselves form two monotone chains, and that the number of all squares of side length  $\ell$  is  $O(n/\ell)$ . The results of the theorem follow by summing up the contributions to the polygon complexity and time complexity for the  $O(\log n)$  different possible values of  $\ell$ .  $\square$

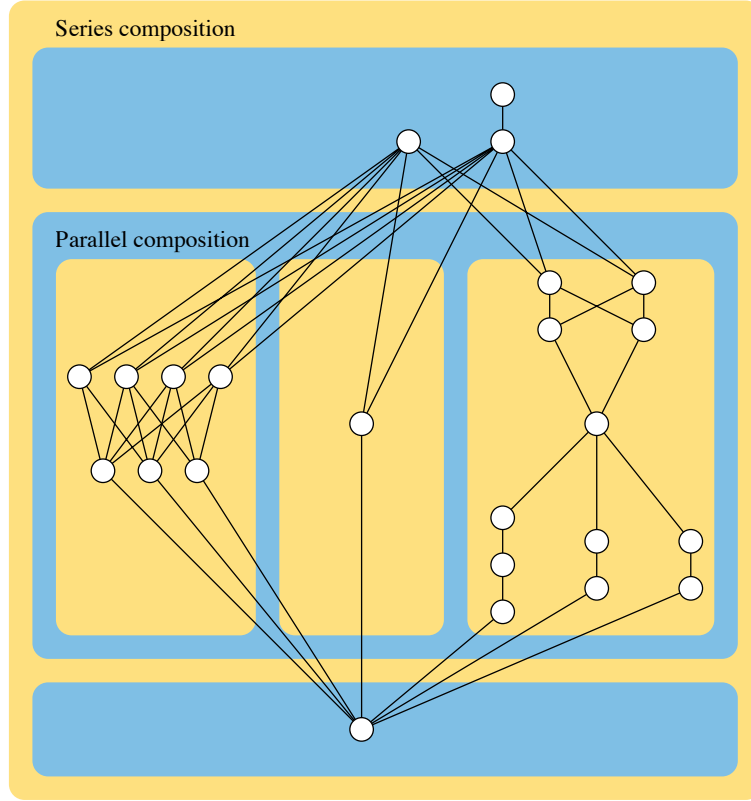
## 4 Series-parallel partial orders

Series-parallel partial orders were considered in the context of a scheduling problem by Lawler [26], and include as a special case the tree orderings previously studied in our work on bicriterion optimization [9].

### 4.1 Recursive decomposition

Although it is possible to characterize the series-parallel partial orders as being the orders in which no four elements form the “N” of Figure 2, it is more convenient for us to define them in terms of a natural recursive decomposition, which we will take advantage of in our algorithms.

**Definition 6.** *The series-parallel partial orders are the partial orders that can be constructed from single-element partial orders by repeatedly applying the following two operations:*



**Fig. 5.** A series-parallel partial order. Redrawn from a Wikipedia illustration by the author.

**Series composition.** Given two series-parallel partial orders  $P_1$  and  $P_2$ , form an order from their disjoint union in which every element of  $P_1$  is less than every element of  $P_2$ .

**Parallel composition.** Given two series-parallel partial orders  $P_1$  and  $P_2$ , form an order from their disjoint union in which there are no order relations between  $P_1$  and  $P_2$ .

See Figure 5 for an example. These composition operations correspond naturally to the two geometric operations on convex polygons that we have already been using.

**Observation 1** If  $P$  is the series composition of  $P_1$  and  $P_2$ , then  $\text{polygon}(P)$  is the convex hull of the union of  $\text{polygon}(P_1)$  and a translate (by the sum of the weights of the elements of  $P_1$ ) of  $\text{polygon}(P_2)$ . If  $P$  is the parallel composition of  $P_1$  and  $P_2$ , then  $\text{polygon}(P)$  is the Minkowski sum of  $\text{polygon}(P_1)$  and  $\text{polygon}(P_2)$ .

Recursively continuing this decomposition gives us a formula for  $\text{polygon}(P)$  in terms of the  $\cup$  and  $\oplus$  operations. By Corollary 1 we immediately obtain:

**Corollary 2.** If  $P$  is a series-parallel partial order with  $n$  elements, then  $\text{polygon}(P)$  has at most  $2n$  vertices.

However, the depth of the formula for  $\text{polygon}(P)$  may be linear, so using Corollary 1 to construct  $\text{polygon}(P)$  could be inefficient. We now describe a faster algorithm. The key idea is to follow the same formula to build  $\text{polygon}(P)$ , but to represent each intermediate result (a convex polygon) by a data structure that allows the  $\cup$  and  $\oplus$  operations to be performed more quickly for pairs of polygons of unbalanced sizes.

## 4.2 Data structure for fast unbalanced polygon merges

Note that a Minkowski sum operation between a polygon of high complexity and a polygon of bounded complexity can change a constant fraction of the vertex coordinates, so to allow fast Minkowski sums our representation cannot store these coordinates explicitly.

**Lemma 6.** *It is possible to store convex polygons in a data structure such that destructively merging the representations of two polygons of  $m$  and  $n$  vertices respectively by a  $\mathbb{U}$  or  $\oplus$  operation (with  $m < n$ ) can be performed in time  $O(m \log((m+n)/m))$ .*

*Proof.* We store the lower and upper hulls separately in a binary search tree data structure, in which each node represents a vertex of the polygon, and the inorder traversal of the tree gives the left-to-right order of the vertices. The node at the root of the tree stores the Cartesian coordinates of its vertex; each non-root node stores the vector difference between its coordinates and its parents' coordinates. Additionally, each node stores the vector difference to its clockwise neighbor around the polygon boundary. In this way, we can traverse any path in this tree and (by adding the stored vector difference) determine the coordinates of any vertex encountered along the path. We may also perform a rotation in the tree, and update the stored vector differences, in constant time per rotation.

We will keep this tree balanced (in an amortized sense) by using the splay tree balancing strategy [39]: whenever we follow a search path in the tree, we will immediately perform a splay operation that through a sequence of double rotations moves the endpoint of the path to the root of the tree. By the dynamic finger property for splay trees [12, 13], a sequence of  $m$  accesses in sequential order into a splay tree of size  $n$  will take time  $O(m \log((m+n)/m))$ .

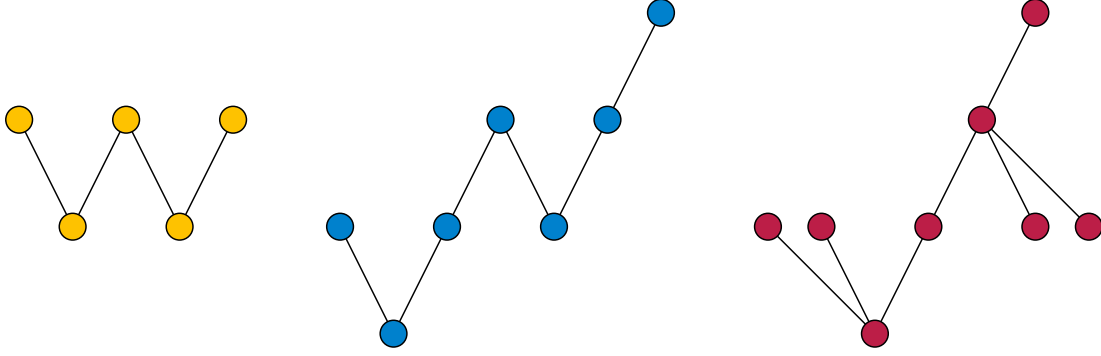
To compute the hull of the union (the  $\mathbb{U}$  operation) we insert each vertex of the smaller polygon (by number of vertices), in left-to-right order, into the larger polygon. To insert a vertex  $v$ , we search the larger polygon to find the edges with the same  $x$ -coordinate as  $v$  and use these edges to check whether  $v$  belongs to the lower hull, the upper hull, or neither. If it belongs to one of the two hulls, we search the larger polygon again to find its two neighbors on the hull. By performing a splay so that these neighbors are rotated to the root of the binary tree, and then cutting the tree at these points, we may remove the vertices between  $v$  and its new neighbors from the tree without having to consider those vertices one-by-one. We then create a new node for  $v$  and add its two neighbors as the left and right child.

To compute the Minkowski sum (the  $\oplus$  operation) we must simply merge the two sequences of edges of the two polygons by their slopes. We search for each edge slope in the smaller polygon. When its position is found, we splay the vertex node at the split position to the root of the tree, and then split the tree into its left and right subtrees, each with a copy of the root node. We translate all vertices on one side of the split by the vector difference for the inserted edge (by adding that vector only to the root of its tree), and rejoin the trees.  $\square$

The proof of the dynamic finger property for splay trees [12, 13] is very complicated, but the same lemma would follow using any other dynamic binary search data structure with the finger property that also allows for an additional operation of removing large consecutive sequences of elements in time proportional to the logarithm of the length of the removed sequence. For instance, level-linked 2-3 trees have the finger property [7] but the proven time bound for deletions in this structure is not of the form we need, so additional analysis would be needed to make them applicable to our problem.

## 4.3 Series-parallel algorithm

With this data structure in hand, we are ready to prove the main result of this section.



**Fig. 6.** A fence (left), generalized fence (center), and polytree (right)

**Theorem 2.** *If  $P$  is a series-parallel partial order, represented by its series-parallel decomposition tree, then  $\text{polygon}(P)$  has complexity  $O(n)$  and may be constructed in time  $O(n \log n)$ .*

*Proof.* We follow the formula for constructing  $\text{polygon}(P)$  by  $\mathbb{U}$  and  $\oplus$  operations, using the data structure of Lemma 6. We charge each merge operation to the partial order elements on the smaller side of each merge. If a partial order element belongs to subproblems of sizes  $n_0 = 1, n_1, \dots, n_h = n$  where  $h$  is the height of the element, then the time charged to it is  $\sum_i O(\log(n_i/n_{i-1})) = O(\log \prod_i (n_i/n_{i-1})) = O(\log n)$ .  $\square$

## 5 Bounded tree-width

In this section, we consider partial orders whose underlying graphs have bounded treewidth. The underlying graph is an undirected graph, obtained by forgetting the edge directions of the covering graph of the partial order. If the partial order is defined by reachability on directed acyclic graphs, the underlying graph of this reachability relation is obtained by forgetting the edge directions of the *transitive reduction*, a minimal directed acyclic graph with the same reachability relation as the given one.

### 5.1 Fences and polytrees

As a warm-up, we first consider the case where the underlying undirected graph has treewidth one: that is, it is a tree or a path. We can bound both the number of solutions to the parametric closure problem and the time for listing all the solutions using a divide-and-conquer approach that we will later generalize to larger values of the treewidth.

**Definition 7.** *A fence or zigzag poset is a partially ordered set whose transitive reduction is a path with alternating edge orientations. A polytree is a directed graph formed by orienting the edges of an undirected tree. An oriented path or up-down poset is a directed graph formed by orienting the edges of an undirected simple path. A generalized fence is a partially ordered set whose transitive reduction is an oriented path. (See Figure 6.)*

Figure 2 depicts an example of a fence. Our definition of a generalized fence follows Munarini [33] and should be distinguished from other authors who define generalized fences as the reachability orders of polytrees [38].

**Theorem 3.** *Let  $P$  be a generalized fence. Then  $\text{polygon}(P)$  has complexity  $O(n^2)$  and can be constructed in time  $O(n^2 \log n)$ .*

*Proof.* We let  $F(n)$  denote the maximum number of vertices of  $\text{polygon}(P)$  where  $P$  ranges over all  $n$ -element generalized fences. To bound  $F(n)$ , let  $P$  be a generalized fence of maximum complexity, and let  $v$  be the middle element of the oriented path from which  $P$  is defined. Let  $L^-$  be the generalized fence determined by the half of the path to the left of  $P$ , after removing all elements below  $v$  in this half of the path, and let  $L^+$  be the generalized fence determined by the left half-path after removing all elements above  $v$ . Define  $R^-$  and  $R^+$  in the same way for the right half of the path. Then the downsets of  $P$  that contain  $v$  correspond one-for-one with downsets of  $L^- \cup R^-$ , where the correspondence is obtained by adding back  $v$  and all elements below  $v$ . The downsets of  $P$  that do not contain  $v$  are exactly the downsets of  $L^+ \cup R^+$ . Therefore, we have the formula

$$\text{polygon}(P) = (\text{polygon}(L^-) \oplus \text{polygon}(R^-)) \uplus (\text{polygon}(L^+) \oplus \text{polygon}(R^+)),$$

with a suitable translation applied before performing the  $\uplus$  operation. Since all four of  $L^-$ ,  $L^+$ ,  $R^-$ , and  $R^+$  have at most  $\lfloor n/2 \rfloor$  elements, we can use this formula and Lemma 1 to derive a recurrence for the complexity  $F(n)$ :

$$F(n) \leq 4F(\lfloor n/2 \rfloor),$$

with the base case  $F(0) = 1$ . This recurrence has a standard form familiar from divide-and-conquer algorithms, whose solution is  $F(n) = O(n^2)$ . Using the same formula for  $\text{polygon}(P)$  together with Lemma 1 to construct the polygon gives another recurrence for the running time  $T(n)$ :

$$T(n) \leq 4T(\lfloor n/2 \rfloor) + O(F(n)) = 4T(\lfloor n/2 \rfloor) + O(n^2) = O(n^2 \log n).$$

□

We may apply the same divide-and-conquer method to construct the solutions to the parametric closure problem for an oriented tree, but we will not always be able to obtain a perfect split into two subproblems of size  $n/2$ . As a consequence, the method becomes somewhat more complex although its asymptotic time bound will remain the same. To pick the splitting vertex  $v$  we use the following separator theorem for trees, which appears already in the work of Jordan (1869) [23]:

**Lemma 7.** *Any  $n$ -vertex undirected tree has a vertex  $v$  such that the removal of  $v$  splits the remaining tree into components of at most  $n/2$  vertices each.*

*Proof.* Let  $v$  be a vertex of the tree  $T$ , chosen to minimize the number  $e$  of vertices by which the largest component of  $T \setminus v$  exceeds  $n/2$ , and suppose for a contradiction that  $e$  is not zero. Let  $C$  be the component of  $T \setminus v$  with  $n/2 + e > n/2$  vertices, and let  $w$  be the unique neighbor of  $v$  in  $C$ . Then removing  $w$  from  $T$  produces components that are strict subsets of  $C$ , together with a component containing  $v$  with  $n/2 - e < n/2$  vertices. Therefore, all components of  $T \setminus w$  have fewer than  $n/2 + e$  vertices, contradicting the choice of  $v$  as the minimizer of  $e$ . This contradiction implies that  $e$  must be zero and therefore that removing  $v$  splits the remaining tree into components of at most  $n/2$  vertices each. □

**Theorem 4.** *Let  $P$  be the reachability order of a polytree  $T$ . Then  $\text{polygon}(P)$  has complexity  $O(n^2)$  and can be constructed in time  $O(n^2 \log n)$ .*

*Proof.* As in Theorem 3 we remove  $v$  from  $T$ , and for each component  $C_i$  of the resulting forest define partial orders  $C_i^-$  and  $C_i^+$  by removing the elements below or above  $C_i$  respectively. And as in Theorem 3 we can use this decomposition to derive a formula

$$\text{polygon}(P) = (\text{polygon}(C_1^-) \oplus \text{polygon}(C_2^-) \oplus \cdots) \uplus (\text{polygon}(C_1^+) \oplus \text{polygon}(C_2^+) \oplus \cdots)$$

By Lemma 1, it follows that the total number of vertices of  $\text{polygon}(P)$  is at most the total number of leaf-level subproblems obtained by recursively performing this decomposition. Because the size of the remaining subtrees goes down by a factor of two in each level of recursion, there are at most  $\log_2 n$  levels of recursion. Each level of recursion also at most doubles the number of subproblems that each remaining tree vertex belongs to: a vertex  $u$  that belongs to a component  $C_i$  of the remaining tree vertices will also be included in at most two subproblems  $C_i^-$  and  $C_i^+$ . Therefore, the contribution of any one tree vertex to the total complexity of  $\text{polygon}(P)$  is  $O(n)$  and the total complexity of  $\text{polygon}(P)$  is  $O(n^2)$ .

We cannot use Lemma 1 directly to evaluate this formula, because that lemma allows only binary combinations. However, using Lemma 6 instead allows any formula of  $\oplus$  and  $\uplus$  operations of size  $s$  to be evaluated in time  $O(s \log s)$ . Applying this method to the recursive formula derived above gives us a total time of  $O(n^2 \log n)$ .  $\square$

In the following sections, we apply the same method to more general graphs of bounded treewidth.

## 5.2 Tree decompositions

**Definition 8.** A tree-decomposition of an undirected graph  $G$  is a tree  $T$  whose vertices are called bags, each of which is associated with a set of vertices of  $G$ , with the following two properties:

- Each vertex of  $G$  belongs to a set of bags that induces a connected subtree of  $T$ , and
- For each edge of  $G$  there exists a bag containing both endpoints of the edge.

The width of a tree-decomposition is one less than the maximum number of vertices in a bag, and the treewidth of  $G$  is the minimum width of a tree-decomposition of  $G$ . By extension, we define the treewidth of a directed graph  $D$  to equal the treewidth of the undirected graph obtained by forgetting the orientations of the edges of  $D$ .

The treewidth, and an optimal tree-decomposition, can be found in an amount of time that is linear in the number of vertices of a given graph but exponential in its width [6].

**Definition 9.** A tree-decomposition of width  $t$  is minimal if no two adjacent bags have a union with at most  $t + 1$  elements.

Every graph of treewidth  $t$  has a minimal decomposition of width  $t$ . For, given a decomposition that is not minimal (in which two adjacent bags have a small union), the edge connecting those two bags could be contracted, and the bags merged, decreasing the number of bags in the decomposition without increasing the width of the decomposition. This property allows us to control the number of bags in the decomposition:

**Lemma 8.** A minimal tree-decomposition of width  $t$  of an  $n$ -vertex graph has at most  $n - t$  bags.

*Proof.* Consider the bags in the order given by a breadth-first traversal, starting from a bag of cardinality  $t + 1$ . The first bag has  $t + 1$  vertices in it, and each successive bag in this traversal must have at least one new vertex not seen in previous bags, for otherwise it would be a subset of the bag that is its parent in the breadth-first traversal and the decomposition would not be minimal. Therefore, the number of vertices is at least equal to the number of bags plus  $t$ . Equivalently, the number of bags is at most  $n - t$ .  $\square$



### 5.3 Partial orders of low treewidth

**Lemma 9.** *Let  $P$  be a partial order with  $n$  elements whose underlying graph has treewidth  $t = O(1)$ . Then  $\text{polygon}(P)$  can be represented as a formula of  $\mathbb{U}$  and  $\oplus$  operations with total size  $O(n^{t+2})$  and depth  $O(\log n)$ .*

*Proof.* We fix a minimal tree-decomposition of width  $t$ , and we use Lemma 7 to recursively partition this decomposition by repeatedly removing a bag  $B$  whose removal splits the remaining part of the decomposition into connected components  $C_i$  of size (number of bags) at most half the size at the previous level of the partition.

If  $B$  has  $b$  elements, then there are at most  $2^b$  partitions of  $B$  into two subsets  $L$  and  $U$ , such that  $L$  is a downset in  $B$  (with  $U$  as its complementary upset). We will use these partitions to specify the elements of  $B$  that should be part of an eventual downset ( $L$ ) or should not be part of an eventual downset ( $U$ ). Let  $C_i$  be one of the connected components formed from the tree-decomposition by the removal of  $B$ , and  $(L_j, U_j)$  be one of the partitions of  $B$  into a downset  $L = L_j$  and an upset  $U = U_j$ . Then in any downset of the whole problem consistent with the partition  $(L_j, U_j)$ , the elements of  $C_i$  that are below at least one element of  $L_j$  must also be included in the downset, and the elements of  $C_i$  that are above at least one element of  $U_j$  must also be included in the upper set. We define  $C_i^j$  to be the subproblem of the closure problem consisting of the remaining elements: the ones that belong to bags of  $C_i$  and are neither below a member of  $L_j$  nor above a member of  $U_j$ .

With this notation, we can express  $\text{polygon}(P)$  as a formula

$$\text{polygon}(P) = \mathbb{U}_{j=1}^{2^b} \oplus_i \text{polygon}(C_i^j),$$

where each term in the outer  $\mathbb{U}$  is translated appropriately according to the weights of the additional elements in or below  $L_j$  that belong to the downsets for that term but not to the subproblems included in that term.

Each of the at most  $\log_2 n$  levels of this recursive partition causes the number of subproblems that each vertex participates in to increase by a factor of at most  $2^{t+1}$ . Therefore, in the whole recursive partition, the total number of subproblems that a single vertex participates in is  $O(n^{t+1})$ . Since there are  $n$  vertices, this partition gives rise to a formula with total complexity  $O(n^{t+2})$ .  $\square$

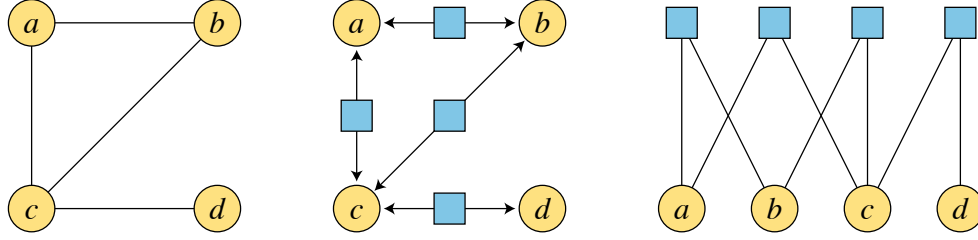
Applying Lemma 6 to the formula of Lemma 9 gives us the following result:

**Theorem 5.** *Let  $P$  be a partial order with  $n$  elements whose underlying graph has treewidth  $t = O(1)$ . Then  $\text{polygon}(P)$  has complexity  $O(n^{t+2})$  and can be constructed in time  $O(n^{t+2} \log n)$ .*

We note that for  $t = 1$  this is larger by a linear factor than the bounds of Theorem 3 and Theorem 4. We do not know whether the dependence on  $t$  in the exponent of the complexity bound is necessary, but unless it can be improved it acts as an obstacle to the existence fixed-parameter-tractable algorithms parameterized by treewidth for the construction of  $\text{polygon}(P)$ .

### 5.4 Incidence posets

The partial orders defined by reachability on graphs of low treewidth include, in particular, the incidence posets of undirected graphs of bounded treewidth. This is because the incidence poset of an undirected graph  $G$  is the same as the reachability poset on a graph obtained from  $G$  by subdividing each undirected edge, and orienting each of the two resulting new edges outward from the subdivision point (Figure 7). This replacement cannot increase the treewidth, as the following folklore lemma shows.



**Fig. 7.** An undirected graph (left), the directed graph obtained by replacing each undirected edge by two oppositely-oriented directed edges (center), and a partial order that is simultaneously the incidence poset of the undirected graph and the reachability poset of the directed graph (right).

**Lemma 10.** *Let  $H$  be obtained from  $G$  by replacing one or more edges of  $G$  by paths. Then the treewidth of  $H$  is no higher than that of  $G$ .*

*Proof.* We may assume that  $H$  is obtained from  $G$  by replacing only a single edge  $e$  by a two-edge path, for any other instance of the lemma may be obtained by multiple replacements of this type. If  $G$  is a tree, then  $H$  is also a tree, and both have treewidth 1. Otherwise, the treewidth of  $G$  is at least 2, and an optimal tree-decomposition for  $G$  must contain at least one bag  $B$  that contains both endpoints of  $e$ . A tree-decomposition for  $H$  of the same width may be obtained by attaching to  $B$  another bag that contains three vertices: the endpoints of  $e$ , and the subdivision point of the replacement path for  $e$ .  $\square$

One of the important applications of the closure problem, to transportation planning, uses the incidence posets of graphs whose edges represent a collection of point-to-point truck routes [4, 37]. Our methods can be applied to the parametric and bicriterion versions of this problem when its graph has small treewidth.

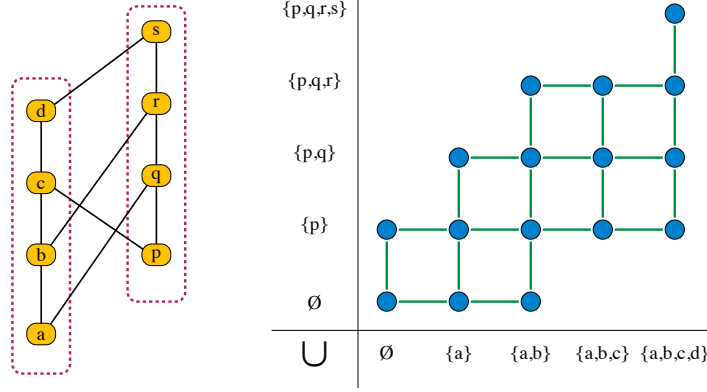
The following result is an immediate corollary of Theorem 5.

**Corollary 3.** *Let  $P$  be the incidence poset of an  $n$ -vertex graph of treewidth  $w$ . Then  $\text{polygon}(P)$  has complexity  $O(n^{w+2})$  and can be constructed in time  $O(n^{w+2} \log n)$ .*

## 6 Bounded width

The *width* of a partially ordered set is the size of its largest antichain. Partially ordered sets of bounded width arise, for instance, in the version histories of a distributed version control repository controlled by a small set of developers (with the assumption that each developer maintains only a single branch of the version history). In this application, there may be many elements of the partially ordered set (versions of the repository) but the width may be bounded by the number of developers [5]. A downset in this application is a set of versions that could possibly describe the simultaneous states of all the developers at some past moment in the history of the repository.

The downsets of a partially ordered set correspond one-for-one with its independent sets: each downset is uniquely determined by an independent set, the set of maximal elements of the downset. Therefore, in a partially ordered set of width  $w$  with  $n$  elements there can be at most  $O(n^w)$  downsets. More precisely, by Dilworth's theorem [14], every partial order of width  $w$  can be partitioned into  $w$  totally-ordered subsets (chains); the number of downsets can be at most the product of the numbers of downsets of these chains. This product takes its maximum value  $(n/w + 1)^w$  when the chains all have equal size. As we show, however, there is an even tighter bound for the complexity of  $\text{polygon}(P)$ .



**Fig. 8.** Hasse diagram of a width-two partial order partitioned into two chains (left) and the subset of the integer lattice formed by its downsets (right)

We consider first the case  $w = 2$ . The more general case of bounded width will follow by similar reasoning. Thus, let  $P$  be a partially ordered set with  $n$  elements and width two. By Dilworth's theorem,  $P$  can be decomposed into two chains, and every downset  $L$  can be described by a pair of integer coordinates  $(x, y)$ , where  $x$  is the number of elements of  $L$  that belong to the first chain and  $y$  is the number of elements of  $L$  in the second chain.

**Observation 2** *For a width-two partial order  $P$  as described above, the set of pairs of coordinates  $(x, y)$  describing the downsets of  $P$  forms an orthogonally convex subset of the integer grid  $[0, n]^2$ , and the edges between points one unit apart in this set of grid points form the covering graph of the distributive lattice of downsets.*

Figure 8 depicts an example. We can use the following definition and observation to partition the parametric closure problem for  $P$  into smaller subproblems of the same type.

**Definition 10.** *Let  $R$  be any grid rectangle. Then  $\text{subproblem}(R)$  is the family of downsets of  $P$  whose grid points lie in  $R$ .*

**Observation 3** *For every grid rectangle  $R$ , each set in  $\text{subproblem}(R)$  can be represented uniquely as a disjoint union  $A \cup B$  where  $A$  is the set of all elements of  $P$  whose (single) coordinate value is below or to the left of  $R$ , and  $B$  is a downset of the restriction of  $P$  to the elements whose coordinate value is within  $R$ .*

In this decomposition, the same set  $A$  forms a subset of every set in  $\text{subproblem}(R)$ . Thus, the observation shows that the contribution to the parametric closure problem from rectangle  $R$  can be obtained by solving a smaller parametric closure problem, on the restriction of the partial order to the elements within  $R$ , and then translating the results by a single vector determined from this fixed set  $A$ .

**Observation 4** *Suppose that every integer point of a grid rectangle  $R$  corresponds to a downset of  $P$ . Then the restriction of  $P$  to the elements whose coordinate value is within  $R$  is a partial order in the form of the parallel composition of two chains, a special case of a series-parallel partial order. It follows from Corollary 2 that  $\text{hull}(\text{project}(\text{subproblem}(R)))$  has complexity proportional to the perimeter of  $R$ .*

**Theorem 6.** *If  $P$  is a partial order of width two, then  $\text{polygon}(P)$  has complexity  $O(n \log n)$  and can be constructed from the covering relation of  $P$  in time  $O(n \log^2 n)$ .*

*Proof.* From the covering relation, we can determine a partition into two chains and trace out the boundaries of the orthogonal grid polygon describing the downsets of  $P$ , in linear time. We use a quadtree to partition the downsets of  $P$  into squares, each with a side length that is a power of two, such that each grid point within each square corresponds to one of the downsets of  $P$ . The squares of each size form two monotone chains within the grid, so the total perimeter of all of these squares is  $O(n \log n)$ .

We also use the one-dimensional projection of the same quadtree, to partition each of the two grid coordinates recursively into subintervals whose sizes are powers of two. For each subinterval  $I$ , we compute the convex hull of the downsets of the corresponding chain whose grid points lie within  $I$ , as the hull of the union of the two previously-constructed polygons for the two halves of  $I$ . This computation takes time  $O(n \log n)$  overall. We also compute a sequence of vectors, the sums of weights of each prefix of the coordinates, in time  $O(n)$ .

We then compute the polygon for each of the squares of the partition of the downsets of  $P$ . By Observation 3, each such polygon can be constructed by translating the polygon for the grid coordinates within the square (the set  $B$  of the observation) by a translation vector determined from all the smaller grid coordinates (the set  $A$  of the observation). By Observation 4, the polygon for the set  $B$  can be computed in time proportional to the perimeter of the square, as the Minkowski sum of the polygons for its two defining subintervals. The translation vector for the set  $A$  can be found as the sum of two vectors computed for the two corresponding prefixes of the two grid coordinates.

The overall solution we seek,  $\text{polygon}(P)$ , can be computed by applying the  $\mathbb{U}$  operation to combine the polygons computed for each square of the partition. As the total perimeter of these squares is  $O(n \log n)$ , and each generates a polygon of complexity proportional to its perimeter in time proportional to its perimeter, the total time to compute all of these polygons is  $O(n \log n)$ , and their total complexity is  $O(n \log n)$ . As the convex hull of  $O(n \log n)$  points, the time to perform the final  $\mathbb{U}$  operation to combine these polygons and compute  $\text{polygon}(P)$  is  $O(n \log^2 n)$ .  $\square$

For higher widths  $w$ , the same idea works (using an octree in three dimensions, etc). The total complexity of  $\text{polygon}(P)$  is proportional to the sum of side lengths of octree squares,  $O(n^{w-1})$ , and the construction time is within a logarithmic factor of this bound.

## 7 Conclusions

We have introduced the parametric closure problem, and given polynomial complexity bounds and algorithms for several important classes of partial orders. Bounds for the general problem, and nontrivial lower bounds on the problem, remain open for further research.

## Acknowledgements

This research was supported in part by NSF grant 1228639 and ONR grant N00014-08-1-1015.

## References

1. P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. *Proc. 39th IEEE Symp. Foundations of Computer Science (FOCS '98)*, pp. 596–605, 1998, doi:10.1109/SFCS.1998.743510.

2. M. Ahmed, I. Chowdhury, M. Gibson, M. S. Islam, and J. Sherrette. On maximum weight objects decomposable into based rectilinear convex objects. *Proc. 13th Int. Symp. Algorithms and Data Structures (WADS 2013)*, pp. 1–12. Springer, Lect. Notes Comput. Sci. 8037, 2013, doi:10.1007/978-3-642-40104-6\_1.
3. P. Avery. An algorithmic proof that semiorders are representable. *J. Algorithms* 13(1):144–147, 1992, doi:10.1016/0196-6774(92)90010-A, MR1146337.
4. M. L. Balinski. On a selection problem. *Manag. Sci.* 17(3):230–231, 1970, doi:10.1287/mnsc.17.3.230.
5. M. J. Bannister, W. E. Devanny, and D. Eppstein. Small superpatterns for dominance drawing. *Proc. Analytic Algorithmics and Combinatorics (ANALCO14)*, pp. 92–103, 2014, doi:10.1137/1.9781611973204.9.
6. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* 25(6):1305–1317, 1996, doi:10.1137/S0097539793251219.
7. M. R. Brown and R. E. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM J. Comput.* 9(3):594–614, 1980, doi:10.1137/0209045, MR584515.
8. K. Buchin, D. Eppstein, M. Löffler, and R. I. Silveira. Adjacency-preserving spatial treemaps. *Proc. 11th Int. Symp. Algorithms and Data Structures (WADS 2011)*, pp. 159–170. Springer, Lect. Notes Comput. Sci. 6844, 2011, doi:10.1007/978-3-642-22300-6\_14.
9. J. Carlson and D. Eppstein. The weighted maximum-mean subtree and other bicriterion subtree problems. *Proc. 10th Scand. Worksh. Algorithm Theory (SWAT 2006)*, pp. 397–408. Springer, Lect. Notes Comput. Sci. 4059, 2006, doi:10.1007/11785293\_37, MR2288327.
10. P. J. Carstensen. Parametric cost shortest path problems. Unpublished memo, Bellcore, 1984.
11. G. J. Chang and J. Edmonds. The poset scheduling problem. *Order* 2(2):113–118, 1985, doi:10.1007/BF00334849, MR815857.
12. R. Cole. On the dynamic finger conjecture for splay trees. II. The proof. *SIAM J. Comput.* 30(1):44–85, 2000, doi:10.1137/S009753979732699X, MR1762706.
13. R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. I. Splay sorting log  $n$ -block sequences. *SIAM J. Comput.* 30(1):1–43, 2000, doi:10.1137/S0097539797326988, MR1762705.
14. R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math.* 51(1):161–166, 1950, doi:10.2307/1969503.
15. D. Eppstein. Geometric lower bounds for parametric matroid optimization. *Discrete Comput. Geom.* 20(4):463–476, 1998, doi:10.1007/PL00009396, MR1651908.
16. B. Faaland, K. Kim, and T. Schmitt. A new algorithm for computing the maximal closure of a graph. *Manag. Sci.* 36(3):315–33, 1990, doi:10.1287/mnsc.36.3.315.
17. D. Fernández-Baca and G. Slutzki. Optimal parametric search on graphs of bounded tree-width. *J. Algorithms* 22(2):212–240, 1997, doi:10.1006/jagm.1996.0816, MR1428337.
18. D. Fernández-Baca, G. Slutzki, and D. Eppstein. Using sparsification for parametric minimum spanning tree problems. *Nordic J. Comput.* 3(4):352–366, 1996, MR1436019.
19. M. Gibson, D. Han, M. Sonka, and X. Wu. Maximum weight digital regions decomposable into digital star-shaped regions. *Proc. 22nd Int. Symp. Algorithms and Computation (ISAAC 2011)*, pp. 724–733. Springer, Lect. Notes Comput. Sci. 7074, 2011, doi:10.1007/978-3-642-25591-5\_74.
20. D. Hochbaum. A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks* 37(4):171–193, 2001, doi:10.1002/net.1012, MR1837196.
21. D. Hochbaum. 50th anniversary article: Selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today. *Manag. Sci.* 50(6):709–723, 2004, doi:10.1287/mnsc.1040.0242.
22. R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the “optimal” stable marriage. *J. ACM* 34(3):532–543, 1987, doi:10.1145/28869.28871, MR904192.
23. C. Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik* 70(2):185–190, 1869, <http://resolver.sub.uni-goettingen.de/purl?GDZPPN002153998>.
24. G. Joret, P. Micek, K. G. Milans, W. T. Trotter, B. Walczak, and R. Wang. Tree-width and dimension, arXiv:1301.5271. Unpublished manuscript, 2013.
25. N. Katoh. Bicriteria network optimization problems. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences* E75-A:321–329, 1992.
26. E. L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math.* 2:75–90, 1978, doi:10.1016/S0167-5060(08)70323-6, MR0495156.
27. H. Lerchs and I. F. Grossmann. Optimum design of open-pit mines. *Trans. Canad. Inst. Mining and Metallurgy* 68:17–24, 1965.
28. M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica* 56(2):235–269, 2010, doi:10.1007/s00453-008-9174-2, MR2576543.
29. M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom. Theory and Appl.* 43(4):419–433, 2010, doi:10.1016/j.comgeo.2009.03.007, MR2575803.
30. R. D. Luce. Semiorders and a theory of utility discrimination. *Econometrica* 24:178–191, 1956, doi:10.2307/1905751, MR0078632.

31. J. Matoušek. Lower envelopes. *Lectures on Discrete Geometry*, pp. 165–194. Springer, Graduate Texts in Mathematics 212, 2002, 10.1007/978-1-4613-0039-7\_7.
32. N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM* 30(4):852–865, 1983, doi:10.1145/2157.322410.
33. E. Munarini. A combinatorial interpretation of the Chebyshev polynomials. *SIAM J. Discrete Math.* 20(3):649–655, 2006, doi:10.1137/S0895480103432283, MR2272221.
34. D. Orlin. Optimal weapons allocation against layered defenses. *Nav. Res. Logist. Q.* 34:605–617, 1987, doi:10.1002/1520-6750(198710)34:5<605::aid-nav3220340502>3.0.co;2-l.
35. J.-C. Picard. Maximal closure of a graph and applications to combinatorial problems. *Manag. Sci.* 22(11):1268–1272, 1976, doi:10.1287/mnsc.22.11.1268, MR0403596.
36. M. Pirlot and P. Vincke. *Semiororders: Properties, Representations, Applications*. Springer, 1997.
37. J. M. W. Rhys. A selection problem of shared fixed costs and network flows. *Manag. Sci.* 17(3):200–207, 1970, doi:10.1287/mnsc.17.3.200.
38. F. Ruskey. Transposition generation of alternating permutations. *Order* 6(3):227–233, 1989, doi:10.1007/BF00563523, MR1048093.
39. D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM* 32(3):652–686, 1985, doi:10.1145/3828.3835.