SIAM J. DISCRETE MATH.
Vol. 16, No. 2, pp. 192–207

MINIMIZING A CONVEX COST CLOSURE SET*

DORIT S. HOCHBAUM† AND MAURICE QUEYRANNE‡

Abstract. Many applications in the area of production and statistical estimation are problems of convex optimization subject to ranking constraints that represent a given partial order. This problem, which we call the convex cost closure (CCC) problem, is a generalization of the known maximum (or minimum) closure problem and the isotonic regression problem. For a CCC problem on n variables and m constraints we describe an algorithm that has the complexity of the minimum cut problem *plus* the complexity of finding the minima of up to n convex functions. Since the CCC problem is a generalization of both minimum cut and minimization of n convex functions, this complexity is the fastest one possible. For the quadratic problem the complexity of our algorithm is strongly polynomial, $O(mn \log \frac{n^2}{m})$. For the isotonic regression problem the complexity is $O(n \log U)$ for U the largest range for a variable value.

Key words. closure problem, nonlinear costs, Bayesian estimation, maximum flow, parametric minimum cut, convex optimization

AMS subject classifications. 68R10, 90C27, 90C30

PII. S0895480100369584

1. Introduction.

A common problem in statistical estimation is that observations do not satisfy preset ranking order requirements. In that case the problem is to find an adjustment of the observations that fits the ranking order constraints and minimizes the total deviation penalty. The deviation penalty is a convex function of the fitted values.

Formally, we define the problem for a directed graph $G = (V, A)$ and a convex function $f_j()$ associated with each node $j \in V$. The formulation of the convex cost closure (CCC) problem is then

$$\begin{array}{lll} \text{(CCC)} \quad \text{Min} & \sum_{j \in V} f_j(x_j) & \\ \text{subject to} & x_i - x_j \geq 0 & \forall (i,j) \in A, \\ & \ell_j \leq x_j \leq u_j \quad \text{integer} & j \in V. \end{array}$$

This problem generalizes the isotonic regression problem in which the graph is a partial order graph for linear order—the arcs of A are of the form $(i, i+1)$. Another well-known problem that CCC generalizes is the minimum closure problem. That problem is the binary case of CCC:

$$\begin{array}{lll} \text{(Minimum Closure)} \quad \text{Min} & \sum_{j \in V} w_j \cdot x_j & \\ \text{subject to} & x_i - x_j \geq 0 & \forall (i,j) \in A, \\ & 0 \leq x_j \leq 1 \quad \text{integer} & j \in V. \end{array}$$

*Received by the editors February 29, 2000; accepted for publication (in revised form) September 18, 2002; published electronically February 20, 2003. An extended abstract of this work appeared in *Algorithms—ESA* 2000, (Saarbrücken), Lecture Notes in Comput. Sci. 1879, Springer, Berlin, pp. 256–267.

http://www.siam.org/journals/sidma/16-2/36958.html

†Department of Industrial Engineering and Operations Research and Walter A. Haas School of Business, University of California, Berkeley, CA 94720 (hochbaum@ieor.berkeley.edu). This author's research was supported by NSF grants DMI-0085690 and DMI-0084857.

‡Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, British Columbia V6T1W5, Canada (quey@commerce.ubc.ca).

Picard established in 1976 [17] that the closure problem is equivalent to a minimum cut problem on a graph associated with G to which we add a source and a sink. This construction is described in section 3. Solving the CCC problem is thus at least as hard as solving the minimum cut problem on the associated graph.

When the graph is empty the CCC problem reduces to the integer minimization of n convex functions, each in a given interval. Thus CCC generalizes the problem of convex functions integer minimization in bounded intervals.

The challenge of the convex optimization problem is that searching for a minimum of a convex function involves an unavoidable factor such as $\log U$ in the running time for U the length of the interval containing the optimal value of the variable. Although one can replace other parameters that depend on the variability of the functions, the running time cannot be made strongly polynomial using the arithmetic complexity model (see [11] for details on this result). The algorithm presented here differs from previous algorithms in that the search for the minima of the convex functions is separate from the rest of the algorithm. The main body of the algorithm identifies disjoint intervals that are guaranteed to contain the optimal values of each variable and satisfy the partial order constraints. The run time of our algorithm, $O(mn \log \frac{n^2}{m} + n \log U)$ for $U = \max_i\{u_i - \ell_i\}$, is the fastest known for the problem, and it either matches or improves the complexity of algorithms devised for special cases of CCC.

In dealing with nonlinear functions it is necessary to specify the complexity model used. We assume the unit cost model and no restriction on the structure of the convex functions; i.e., we assume the existence of an oracle returning function values for every polynomial length argument input in $O(1)$. Since we will search for an optimal solution among the integers we will be interested only in integer arguments. Any arithmetic operation or comparison involving function values is executed in unit time in this model. Derivatives, or rather subgradients, are required as finite differences, $f_j'(x) = f_j(x+1) - f_j(x)$.

In the next section we give an overview of the literature and applications of the problem. In section 3 the link of the closure problem to the maximum flow problem is reviewed. Section 4 discusses a linear time algorithm that is employed to verify whether an instance of CCC is feasible. Section 5 provides the main theoretical underpinning of the algorithm, the so-called *threshold theorem*. Section 6 describes the entire algorithm and its correctness and complexity. Section 7 details the implementations in strongly polynomial time for the quadratic case and the $O(n \log U)$ for the isotonic regression problem. In Section 8 we provide an algorithm for the continuous version of CCC. In Section 9 we conclude with several remarks and extensions.

2. Related applications and literature. We sketch first several classes of applications for the CCC problem.

In the problem of *selection of discrete contingent projects* a number of projects $j \in N$ can be undertaken, but only at discrete levels $l_j, l_j + 1, \ldots, u_j$. These projects are contingent in that, for specified pairs of projects $(i, j) \in A$, each unit of project i requires one unit of project j: $x_i \leq x_j$. Different projects i, k, ..., however, can use the same units of project j (otherwise, we might consider j as part of project i). For instance, projects i and k may use j at different times. The objective is to maximize the total net profit associated with the selected project levels. Here, $-f_j(x_j)$ denotes the net profit associated with level x_j of project j. The convexity of $-f_j$ thus reflects decreasing returns of scale for project j. Additional information about this problem is provided by Picard and Queyranne [18].

Maxwell and Muckstadt [16] considered *nested power-of-two policies* in a *multi-stage production/inventory* problem. In this continuous-time deterministic model, demand for end-products arises at a constant rate. Intermediate products are consumed in the production of other products, as reflected by the directed graph (V, A). Given are positive inventory-related holding costs g_j and production setup costs K_j. The problem is to find production intervals $T_j = T_0 2^{k_j}$, with k_j integer, that are *nested*, that is, $T_i \leq T_j$ for $(i, j) \in A$. The objective is to minimize the average total cost per unit time,

$$c(T) = \sum_j g_j T_j + K_j / T_j.$$

Roundy [20] extends the Maxwell–Muckstadt model by considering joint setup costs and relaxing the nestedness condition. He shows that the total cost is now

$$c(T) = \sum_R G_R \max\{T_j : j \in R\} + \sum_F K_F / (\min\{T_j : j \in F\}),$$

where R and F are suitably defined subsets of products, and G_R and K_F are corresponding (nonnegative) costs. Although the constraints $T_i \leq T_j$ thus disappear, the modeling capabilities of variable upper bound constraints are reflected in the handling of joint setup costs and holding costs. For that, Roundy extends the product set N by adding all the R and F sets and "defines" corresponding variables T_R (T_F, respectively) by the inequalities $T_j \leq T_R$ ($T_F \leq T_j$, respectively) for all $j \in R$ (F, respectively). The resulting problem is thus recast into the Maxwell–Muckstadt form above. Roundy's major result is that *optimal power-of-two policies* thus constructed are 94% *effective*; that is, the cost of an optimal policy cannot be less than 94% of an optimum power-of-two policy. He also shows that searching for an optimal base interval T_0 yields a 98% *effective* solution. The present paper extends this approach to general convex average cost functions $f_j(k_j) = c_j(T_0 2^{k_j})$. The 94% and 98% effectiveness results, however, hold only for the specific functions c above.

Sokkalingam, Ahuja, and Orlin [23] discuss the *inverse spanning tree* problem. In this problem there is a spanning tree T given in an edge weighted graph. The problem is to modify the edge weights so that the given tree is a minimum spanning tree and the cost of the deviation is minimum. In order for a tree to be a minimum spanning tree each out-of-tree edge j must have a weight w_j greater than or equal to each of the edges in the tree on the unique path between its endpoints. That is, the constraints enforcing that T be a minimum spanning tree are of the form $w_j \geq w_i$ for $j \in E \setminus T$, $i \in T$. The corresponding graph is a bipartite graph—a structure that can be used to reduce the complexity of our algorithm for the resulting CCC. Sokkalingam, Ahuja, and Orlin devised algorithms for three specific convex deviation functions: sum of absolute differences, weighted sum of absolute differences, and maximum absolute differences. All these functions are convex for which the minima can be found in a single step. Our algorithm's run time is thus strongly polynomial and is better than $O(mn \log^2 n)$ for this type of function, and with an additional additive factor of $n \log C$ for general convex functions, where C is the maximum edge weight. The complexity reported in [23] for the weighted absolute deviation problem is weakly polynomial $O(n^2 m \log(nC))$. Our algorithm can be further adapted to provide substantial improvements for special cases as reported in [14].

Statistical problems of *partially ordered estimation* have been discussed extensively in the literature; see, e.g., Veinott [24] and Barlow et al. [4]. Let $p_1, \ldots, p_n$

denote parameters to be jointly estimated and let $f_j(x_j)$ denote the *loss* associated with estimating that $p_j = x_j$ for $j = 1, \ldots, n$. A typical instance is when $f_j(x_j)$ is the negative of the logarithm of the likelihood, given $p_j = x_j$, that related random variables assume observed values. The model being estimated may specify a *partial order* on the parameters, as reflected by constraints $x_j \leq x_i$ for a set A of pairs (i, j), as well as simple upper and lower bounds on the parameter values. If, in addition, the model requires the parameter values to be integer, then the problem of jointly estimating the parameter values so as to minimize total loss is precisely an instance of the CCC problem. If there is no such integrality restriction, then the problem is an instance of the continuous relaxation of CCC, which is discussed in section 8.

Algorithms for the CCC problem have been previously devised. Picard and Queyranne [19] proposed an algorithm solving the problem with a running time of $O(n(mn \log \frac{n^2}{m} + n \log U))$. Ahuja, Hochbaum, and Orlin [3] addressed a generalization of CCC—a convex cost dual of minimum cost network flow. Their algorithm for this convex cost dual of minimum cost flow has running time of $O(mn \log \frac{n^2}{m} \log(nU))$.

The method of Hochbaum and Naor [9] solves integer problems on monotone inequalities in at most two variables per inequality. A monotone inequality is of the form $ax - by \leq c$, where the coefficients of x and y are of opposite signs. Obviously, the constraints of CCC are monotone inequalities. The algorithm of Hochbaum and Naor runs in pseudopolynomial time $O(\sum_i (u_i - \ell_i) mU \log \frac{n^2 U}{m})$. It is possible to combine that algorithm with a scaling approach implemented for CCC in time $O(mn \log U \log \frac{n^2 U}{m})$; see [2]. Hochbaum [12] generalized the concept of monotone inequality to include three variables, $ax - by \leq c + z$, for $a, b \geq 0$. The run time for solving integer programming on such inequalities was shown in [12] to be solved in the same time as the algorithm in [2].

The algorithm described here solves CCC in time $O(mn \log \frac{n^2}{m} + n \log U)$. The first term in the complexity expression is the run time required to solve the minimum closure problem, and the second factor is the run time required to find the integer minima of n convex functions. Since CCC generalizes both these problems, as discussed above this is the best complexity achievable for CCC. It is likely that if a faster algorithm for the minimum closure problem is discovered, then the run time of the algorithm can be respectively improved. For the second term the factor of $\log U$ cannot be avoided, as any algorithm solving a constrained nonlinear and nonquadratic optimization problem may not run in strongly polynomial time [11]. When the functions f_i are quadratic convex, the algorithm runs in strongly polynomial time $O(mn \log \frac{n^2}{m} + n \log n)$. For the isotonic regression problem the running time improves to $O(n \log n + n \log U)$, and thus the complexity of our algorithm is $O(n \log(\max\{n, U\}))$.

There are efficient algorithms known for solving several special cases of CCC. Any maximum flow algorithm can be used to solve the minimum (or maximum) closure problem. The most efficient algorithm known to date, due to Goldberg and Tarjan [8], solves the maximum flow and thus the minimum cut, and the closure problems have complexity of $O(mn \log \frac{n^2}{m})$.

The isotonic regression problem is an instance of CCC defined on a linear order. Ahuja and Orlin report on an $O(n \log U)$ time algorithm for this problem [1]. The problem has been reviewed extensively in the statistical study of observations. Barlow et al. [4] provide an excellent review of applications and algorithms for the isotonic regression problem as well as the CCC problem.

3. Solving the minimum closure as a minimum cut problem. Recall that the minimum closure problem is a special case of CCC attained by setting the variables to be binary. We review here the procedure for solving the minimum closure problem with a minimum cut algorithm. Although CCC is a problem far more general than the minimum closure, our algorithm for CCC also solves the minimum closure problem in the most efficient complexity known.

A set of nodes $S \subseteq V$ in a directed graph $G = (V, A)$ is said to be *closed* if all predecessor nodes of S are also included in S; i.e., if $j \in S$ and $(i, j) \in A$, then $i \in S$. Equivalently, S is said to be closed if it has no incoming arcs.

We review here the reduction of Picard [17], demonstrating that the minimum closure problem is solved using a minimum cut procedure. We first define an s, t-graph that contains a source and a sink and that is associated with the minimum closure problem: the graph has a node j associated with each variable x_j. A source, and sink nodes s and t, are now added to the graph. If the weight of the variable w_j is positive, then node j has an arc from the source into it with capacity w_j. If the node has weight w_j which is negative, then there is an arc from j to t with capacity $-w_j$. Let V^+ be the set of nodes with positive weights and V^- be the set of nodes with negative weights.

Each inequality $x_i \geq x_j$ is associated with an arc (i, j) of infinite capacity. Consider any finite s, t-cut in the graph that partitions the set of nodes into two subsets commonly referred to as the *source set* of the cut and the *sink set* of the cut, $\{s\} \cup S$ and $\{t\} \cup \bar{S}$. It is easy to see that $\bar{S}$ is a closed set if there are no infinite capacity arcs from S to $\bar{S}$.

We denote by (A, B) the collection of arcs with tails at A and heads at B. The corresponding sum of capacities of these arcs is denoted by $C(A, B)$, $C(A, B) = \sum_{i \in A, j \in B} c_{ij}$, where c_{ij} is the capacity of arc (i, j) . Let $w(A) = \sum_{j \in A} w_j$.

Given a finite cut $(\{s\} \cup S, \bar{S} \cup \{t\})$, we have

$$\begin{aligned} \min_{\bar{S} \subseteq V} [C(\{s\} \cup S, \bar{S} \cup \{t\})] &= \min_{\bar{S} \subseteq V} \sum_{j \in \bar{S} \cap V^+} w_j + \sum_{j \in S \cap V^-} (-w_j) \\ &= \min_{\bar{S} \subseteq V} \sum_{j \in \bar{S} \cap V^+} w_j - (\sum_{i \in V^-} w_i - \sum_{i \in \bar{S} \cap V^-} w_i) \\ &= \min_{\bar{S} \subseteq V} \sum_{j \in \bar{S}} w_j - w(V^-). \end{aligned}$$

In the last expression the term $w(V^-)$ is a constant. Thus the closed set $\bar{S}$ of minimum weight is also the sink set of a minimum cut and vice versa—the sink set of a minimum cut (without t), which has to be finite, also minimizes the weight of the closure.

4. Verifying feasibility in linear time. We define a graph associated with CCC that has one node representing each variable in the problem. We let the set of nodes be denoted by V. Each inequality $x_i \geq x_j$ is associated with an arc (i, j). We let the set of arcs be denoted by A. If the directed graph (V, A) has strongly connected components, then each node in the strongly connected component shares a directed cycle with each of the other nodes in the strongly connected component, and thus the values of the corresponding variables must be equal.

Finding the strongly connected components of a graph can be accomplished in $O(m)$ time; see, e.g., [6, Chap. 23]. The strongly connected components partition the nodes of the graph into $V_1 \cup \cdots \cup V_k$. In each strongly connected component V_i we let $\ell(V_i)$ be the tightest lower bound in V_i and let $u(V_i)$ be the tightest upper bound in V_i. That is,

$$\ell(V_i) = \max_{v \in V_i} \ell_v \quad \text{and} \quad u(V_i) = \min_{v \in V_i} u_v.$$

A necessary condition for feasibility is that all variables in the same strongly connected component assume the same value that falls in the interval range $[\ell(V_i), u(V_i)]$. Since the above recursion is performed in linear time, verifying this necessary condition amounts to checking that $\ell(V_i) \leq u(V_i)$ in $O(m+n)$ steps.

We now consolidate each strongly connected component into a single node V_i defined on the interval $[\ell(V_i), u(V_i)]$. The function associated with such a node (or variable) is the sum of the convex functions associated with all nodes in V_i, which is a convex function. The graph of strongly connected components is thus a directed acyclic graph (DAG).

Let V_i be a predecessor of V_j in different strongly connected components. Then the following updates are valid:

$$\ell(V_i) \leftarrow \max\{\ell(V_j), \ell(V_i)\}, \qquad u(V_j) \leftarrow \min\{u(V_i), u(V_j)\}.$$

All these updates can be performed in time $O(m)$. A necessary condition for feasibility is that for $i = 1, \ldots, k$, $\ell(V_i) \leq u(V_i)$. This condition is also sufficient since, if satisfied, there exists a feasible solution which is, say, to set all variables to the lower bounds of their corresponding intervals.

Our optimization algorithm runs faster if the feasibility preprocessing step is performed and the interval bounds are adjusted. This preprocessing step, however, is not essential and does not affect the worst case complexity.

5. The threshold theorem. The threshold theorem is the cornerstone of our algorithm. The theorem is an extension of an earlier result of Picard and Queyranne [19].

Let α be a scalar in the interval $(\ell, u) = (\min_{i \in V} \ell_i, \max_{i \in V} u_i)$. Consider further the convex extension of the functions $f_i()$ on the real line by setting $f_i'(x)$ to be equal to M at values of $x > u_i$, and to $-M$ for values $x < \ell_i$, for M a suitably large value. We will comment after the statement of the theorem on how large M should be. The functions $f_i()$ are therefore defined for any real value x as follows:

$$f_i(x) = \begin{cases} f_i(u_i) + M(x - u_i) & \text{if } \; x > u_i, \\ f_i(x) & \text{if } \; \ell_i \leq x \leq u_i, \\ f_i(\ell_i) + M(\ell_i - x) & \text{if } \; x < \ell_i. \end{cases}$$

Consider now the minimum closure problem with variable weights $w_i = w_i(\alpha)$ that are the subgradients of f_i at α, $w_i = f_i'(\alpha) = f_i(\alpha + 1) - f_i(\alpha)$. The theorem establishes that all elements i in the minimum weight closed set S^* satisfy that for any optimal solution $\mathbf{x}$, $x_i > \alpha$, and satisfy that for all elements j in the complement of S^*, $x_j \leq \alpha$. Consequently, the theorem allows for the reduction of CCC to a sequence of minimum closure problems.

In case there are several optimal minimum closed sets, we define a *minimal* minimum closed set as a minimum closed set that does not contain other minimum closed sets. Similarly, a *maximal* minimum closed set is defined as a minimum closed set that is not contained in another minimum closed set.

THEOREM 5.1. *Let $w_i = f_i'(\alpha)$ be the weight assigned to node i, $i = 1, \ldots, n$, in a minimum closure problem defined on the directed graph $G = (V, A)$. Let S^* be the minimal minimum weight closed set in this graph. Then an optimal solution $\mathbf{x}^*$ to the CCC problem satisfies $x_i^* > \alpha$ if $i \in S^*$ and $x_i^* \leq \alpha$ if $i \in \bar{S}^*$.*

Proof. The proof is by contradiction. Let S^* be the minimal minimum weight closed set, and suppose there is a nonempty subset $S^o \subseteq S^*$ such that at an optimal solution $\mathbf{x}^*$, $x_j^* \leq \alpha$ for all $j \in S^o$.

Since at the optimum $x_j^* > \alpha$ for $j \in S^* \setminus S^o$, the set $S^* \setminus S^o$ must be closed, as it has no predecessors (larger values) in S^o. But this set is not a minimal minimum closed set, as S^* is minimal. Thus the weight of nodes in S^o—the total sum of subgradients—must be negative, $\sum_{j \in S^o} f_j'(\alpha) < 0$. Furthermore, increasing the values of all x_j^* in this set to $\alpha + \epsilon \leq \min_{i \in S^* \setminus S^o} x_i^*$ for some $\epsilon > 0$ does not violate feasibility, since the values of their predecessors in $S^* \setminus S^o$ are all $\geq \alpha + \epsilon$. Thus replacing x_j^* for $j \in S^o$ by α is feasible and strictly reduces the weight of the closure compared to an optimal solution. This contradicts the assumption that $\mathbf{x}^*$ is optimal.

An analogous contradiction is reached if we assume that an optimal solution has in the set $\bar{S^*}$ a variable with value $> \alpha$. □

As a result of the theorem, we can decompose the set of nodes into subsets that imply a narrowing of the interval in which the optimal value of the respective variable is to be found: For a given value of α we solve the minimum closure problem with $w_i = f_i'(\alpha)$ for a minimal minimum closure S^*. For all $i \in S^*$ we conclude that $x_i^* \in (\max\{\ell_i, \alpha\}, u_i]$ and for all $j \in \bar{S^*}$, $x_j^* \in [\ell_j, \min\{\alpha, u_j\}]$.

Concerning the value of M, it is sufficient to set $M = \sum_i \max\{f_i'(u_i), |f_i'(\ell_i)|\}$. We claim that for a feasible problem a node with weight M is never in a minimum weight closed set, and a node with weight $-M$ is always in a minimum weight closed set. If that were not the case, then either we can generate a closed set of a strictly lower value by including nodes of weight $-M$ and excluding nodes of weight M or else there is a node j of weight $-M$ that has as its predecessor a node i of weight M. But that means that the given value of α satisfies $u_i < \alpha < \ell_j$, and there is no feasible solution where the value of $x_i \in [\ell_i, u_i]$ is at least as large as the value of $x_j \in [\ell_j, u_j]$.

Indeed, the algorithm we employ to solve CCC can be used to verify feasibility as well—for every value of α the nodes of weight M must be in the source set and the nodes of weight $-M$ must be in the sink set or else the problem is infeasible. Yet, if the feasibility test of section 4 is used, then whenever the threshold theorem is invoked in the algorithm the nodes of weight $-M$ are known a priori to be in the minimum closure and thus in the sink set, and those nodes of weight M are known to be in the source set. This permits the "shrinking" of nodes of weight M with the source and nodes of weight $-M$ with the sink. The size of the graph is thus reduced and the value of M is not explicitly used if the feasibility test is invoked as a preprocessing step.

6. The algorithm. One obvious method of using the threshold theorem for solving CCC is to perform a search by calling for the solution of the minimum closure problem for all integer values of α in the interval (ℓ, u). When done, the output of such a process is a partition of the set of variables V into q sets, and the interval into q disjoint intervals, so that all variables in the same set have their optimal values in the same interval. The goal would be to find for each variable x_j the largest value of α for which it is still in the source set and to find the smallest value of α for which it is no longer in the source set. With this information we narrow down the value of x_j at an optimal solution to an interval defined by these values. We later show that once these intervals are identified, all variables assigned to the same interval assume the same value in that interval, and that value is the lower end of the interval. One drawback of the approach just described is that it makes U calls to a minimum cut procedure and is thus pseudopolynomial.

It is easy to see that a binary search type approach could be used to implement the procedure of identifying the intervals to a polynomial time procedure. Next we show that one can do still better by implementing the process of identifying the set

and interval partitioning in strongly polynomial time and in the complexity of solving a single minimum cut problem.

The key to our approach is to utilize *parametric minimum cut* to generate all the breakpoints of the decompositions. This can be done, as is shown here, by adapting the method of Gallo, Grigoriadis, and Tarjan [7], which works in the same running time as a single minimum cut procedure.

6.1. The parametric graph G_λ. We create a graph with parametric capacities, $G_\lambda = (V \cup \{s,t\}, A)$. Each node $j \in V$ has an incoming arc from s with capacity $\max\{0, f_j'(\lambda)\}$ and an outgoing arc to the sink t with capacity $-\min\{0, f_j'(\lambda)\}$. The capacities of the arcs adjacent to the source in this graph are monotone nondecreasing as a function of λ, and the arcs adjacent to the sink have capacities that are monotone nonincreasing as a function of λ. Note that each node is connected with a positive capacity arc, either to source, or to sink, but not to both. Denote the source set of a minimum cut in the graph G_λ by S_λ.

Restating the threshold theorem in terms of the corresponding minimum cut for the graph G_λ associated with the closure graph, any optimal solution $\mathbf{x}$ satisfies that $x_j > \lambda$ for $j \in \bar{S}_\lambda$ and $x_j \leq \lambda$ for $j \in S_\lambda$, where S_λ is the maximal source set of a minimum cut.

Let ℓ be the lowest lower bound on any of the variables and u the largest upper bound. Consider varying the value of λ in the interval $[\ell, u]$. As the value of λ increases, the sink set becomes smaller and contained in the previous sink sets corresponding to smaller values of λ, specifically, for some $\lambda \leq \ell$ $S_\lambda = \{s\}$ and some $\lambda \geq u$ $S_\lambda = V \cup \{s\}$. We call each value of λ, where S_λ strictly increases, a *node-shifting breakpoint*. For $\lambda_1 < \cdots < \lambda_\ell$ the set of all node-shifting breakpoints we get a corresponding nested collection of source sets,

$$\{s\} = S_{\lambda_1} \subset S_{\lambda_2} \subset \cdots \subset S_{\lambda_\ell} = \{s\} \cup V.$$

Our goal is to partition the variables into the subsets $S(k) = S_{\lambda_k} - S_{\lambda_{k-1}}$, $k = 2, \ldots, \ell$. The property of each subset $S(k)$ is that all variables in the set have optimal value in the interval $(\lambda_{k-1}, \lambda_k]$. As we prove next, the optimal value of all variables in $S(k)$ is x^*, where

$$x^* = \lambda_{k-1} + 1.$$

LEMMA 6.1. *For $j \in S(k)$, the value of x_j at an optimal solution, x_j^*, is $\lambda_{k-1}+1$.*

Proof. According to the threshold theorem, λ_{k-1} is the largest value so that for $j \in S(k)$, $x_j > \lambda_{k-1}$.

It follows that $x_j = \lambda_{k-1} + 1$. ∎

6.2. Identifying an integer node-shifting breakpoint. Since we are interested only in integer valued solutions, we can consider the convex functions $f_j(x)$ to be piecewise linear segments connecting the values of $f_j(k)$ on integer points $\ell_j \leq k \leq u_j$. For such functions the derivatives at the integer points are not well defined and indeed could be any subgradient of the function at the respective integer point. We will consider the derivative $f_j'(x)$ to be a step function with the value in the interval $(k-1, k]$ equal to $f_j(k) - f_j(k-1)$.

We denote a maximal minimum cut source set in G_λ by $S_\lambda^{\max}$ and a minimal minimum cut source set by $S_\lambda^{\min}$.

The source set of a minimum cut of G_λ remains invariant for $\lambda \in (k-1, k]$. Thus, in order to verify that λ is a node-shifting breakpoint, it suffices to compare $S_\lambda^{\max}$

with $S^{\min}_{\lambda+\epsilon}$ for $\epsilon > 0$ sufficiently small. In our case we consider only integer values of λ, and $\epsilon = 1$ is a small enough value. So if $S^{\max}_{\lambda} \subset S^{\min}_{\lambda+1}$, then λ is a node-shifting breakpoint.

The existence of a breakpoint in an interval (λ_1, λ_2) is confirmed if and only if $S^{\max}_{\lambda_1} \subset S^{\min}_{\lambda_2}$.

6.2.1. Parametric analysis. Gallo, Grigoriadis, and Tarjan [7] devised a complete parametric analysis algorithm using the push-relabel algorithm that runs in the same time as a single push-relabel algorithm and identifies all node-shifting breakpoints. The algorithm is applicable to graphs with source adjacent arcs having capacities monotone nondecreasing in the parameter λ and sink adjacent arcs having capacities nonincreasing in λ. The running time of the algorithm for linear capacity functions is $O(mn \log \frac{n^2}{m})$. The same result is achieved using the pseudoflow algorithm [13] with a running time of $O(mn \log n)$. We let the generic run time be Qmn, where Q is a constant times $\log \frac{n^2}{m}$ for push-relabel and a constant times $\log n$ for pseudoflow. Whenever we refer in the analysis below to a minimum cut algorithm it can be either the push-relabel algorithm or the pseudoflow algorithm (and its variants). Other minimum cut algorithms do not satisfy the necessary requirements to make them amenable to the analysis of the parametric procedure.

We assume henceforth that the procedure **min-cut**(G_λ) returns both the minimal and maximal source sets of minimum cuts (if different), $S^{\min}_{\lambda}$, $S^{\max}_{\lambda}$, and $S^{\min}_{\lambda+1}$. The procedure also returns the state of the graph at the end of the run, which includes node labels and preflows for the push-relabel algorithm and node labels and pseudoflows for the pseudoflow algorithm.

For a given interval (λ_1, λ_2) we can find all node-shifting breakpoints by using the procedure **parametric**. The input to the procedure includes R_1 and R_2, which are runs of the minimum cut algorithm that are initiated on an s,t-graph G and the reverse graph G^R, respectively.

Procedure parametric $(G, \lambda_1, \lambda_2, S^{\max}_{\lambda_1}, S^{\min}_{\lambda_2}, R_1, R_2)$.
Contract in G: $s \leftarrow s \cup S^{\max}_{\lambda_1}$, $t \leftarrow t \cup \bar{S}^{\min}_{\lambda_2}$. If $V = \{s, t\}$, or, if $\lambda_2 - \lambda_1 \leq 1$, halt
"no breakpoints."
Else, let $\lambda^* = \lfloor \frac{\lambda_1+\lambda_2}{2} \rfloor$.
Call **min-cut**$(G_{\lambda^*}, R_1, R_2)$ for the output $S^{\min}_{\lambda^*}$, $S^{\max}_{\lambda^*}$, and R^*.
If λ^* is a breakpoint, output λ^* and $S^{\min}_{\lambda^*}$.
Call **parametric** $(G, \lambda_1, \lambda^*, S^{\max}_{\lambda_1}, S^{\min}_{\lambda^*}, R_1, R^*)$.
Call **parametric** $(G, \lambda^*, \lambda_2, S^{\max}_{\lambda^*}, S^{\min}_{\lambda_2}, R^*, R_2)$.
end

The choice of λ^* as the median in the interval (λ_1, λ_2) leads to an additional run time of $O(n \log(\lambda_1 - \lambda_2))$, where n is the number of adjusted capacity functions. For specific capacity functions λ^* is replaced by the intersection of the two cut capacity functions.

The analysis of the complexity of the procedure follows arguments used in [7].[1] It is essential that the algorithm used in the runs for minimum cut satisfies the following properties:

[1] The source of some of this analysis is from private communication of the first author with R. Tarjan in 1996.

- *Reflectivity*: The complexity of the algorithm remains the same whether run on the graph or reverse graph.
- *Monotonicity*: Running the algorithm on a monotone sequence of parameter values has the same complexity as a single run.

The main recursive procedure is **min-cut**$(G_{\lambda^*}, R_1, R_2)$, where R_1 is the status of the graph (labels assigned to nodes and flow values) G_{λ_1} after a minimum cut was identified and R_2 is the state of the graph after the minimum cut was found on the reverse graph $G^R_{\lambda_2}$.

The procedure is implemented as follows: Run a maximum flow algorithm on G_{λ^*} as a monotone continuation of the run R_1. Concurrently, run a maximum flow algorithm on $G^R_{\lambda^*}$ as a monotone continuation of the run R_2. Suppose that the algorithm for the forward direction (on G) stops first (the other case is symmetric). If $|S^{\min}_{\lambda^*}| > n/2$, complete the execution of the maximum flow algorithm on G^R and let R^* be the resulting state of the graph.

Consider the execution that follows immediately of the recursive calls to **parametric** $(G, \lambda_1, \lambda^*, S^{\max}_{\lambda_1}, S^{\min}_{\lambda^*}, R_1, R^*)$, and to **parametric** $(G, \lambda^*, \lambda_2, S^{\max}_{\lambda^*}, S^{\min}_{\lambda_2}, R^*, R_2)$. Consider graphs $G(\bar{S}^{\min}_{\lambda^*})$ and $G(S^{\max}_{\lambda^*})$ on which **min-cut** is called recursively. Let R_3 and R_4, respectively, be the forward and backward runs on $G(\bar{S}^{\min}_{\lambda^*})$ when **min-cut** is applied. Let R_5 and R_6, respectively, be the forward and backward runs on $G(S^{\max}_{\lambda^*})$ when **min-cut** is applied. We distinguish two cases.

Case 1. If $n_1 > n/2$, we regard R_4 as a continuation of R_2, and regard R_3 as a restart of R_1, that is, as a continuation of the run of which R_1 was a continuation. We must charge for R_5 and R_6 as starts of new runs. The $2Qm_1n_1$ term in the recurrence for $T(m, n)$ below accounts for the new runs of the push-relabel algorithm that begin with R_5 and R_6.

Case 2. Symmetrically, if $n_1 \leq n/2$, we regard R_5 as a continuation of R_1, and regard R_6 as a restart of R_2. In this case, the $2Qm_1n_1$ term in the recurrence for $T(m, n)$ below accounts for the new runs that begin with R_3 and R_4.

In Case 1, we still must account for the cost of R_1. In Case 2, we still must account for the cost of R_2. Procedure **min-cut** runs R_1 and R_2 concurrently, stopping when the first one stops. Suppose R_1 stops first. Then the cost of R_1 is covered by the cost of R_2, which takes care of Case 1. Note that in this case R_2 is run to completion, even though it takes longer than R_1 (see implementation); R_1 is the abandoned run, but it is cheaper than R_2, which is the good run. Suppose R_1 stops first but we are in Case 2. Although run R_2 is abandoned, we have spent no more time on it than the time spent running R_1, which was a good run. In this case, the run of R_1 covers the time spent on (partially) running R_2. The situation is symmetric if R_2 stops first. In every case the time spent on the completed good run is at least as much as the time spent on partially or completely performing the run that is abandoned.

Throughout the procedure the total complexity of abandoned runs is at most the complexity of one run with monotonically increasing (or decreasing) parameter values. In addition, the total work for good runs on G_{λ^*} and $G^R_{\lambda^*}$ is at most twice the complexity of one run on monotone parameter values. The total complexity charged for these runs is at most that of three runs of the minimum cut algorithm, $3Qmn$.

Let $m_1 + m_2 \leq m$, $n_1 + n_2 \leq n$, and $n_1 \leq \frac{1}{2}n$. The running time $T(m, n)$ is the additional running time required by the algorithm, taking into account the new runs initiated with each recursive call to **min-cut**. Let Q be a constant. Then

$$T(m, n) = T(m_1, n_1) + T(m_2, n_2) + 2Qm_1n_1.$$

The solution to the recursion is $T(m,n) = Qmn$. Thus the overall run time of the parametric procedure with the push-relabel algorithm is $O(mn \log \frac{n^2}{m})$. The run time incurred in adjusting capacities is $O(n \log U)$ throughout the procedure.

6.3. The algorithm. Let ℓ be the lowest lower bound on any of the variables and u be the largest upper bound. Let $U = u - \ell$.

PROCEDURE CONVEX COST CLOSURE $(G, f_j,\ j = 1, \ldots, n)$.

Step 1: Call **parametric** $(\ell, u, \emptyset, V)$.

Let the output be a set of up to n breakpoints $\lambda_1, \lambda_2, \ldots, \lambda_\ell$ and the corresponding sets of source sets of minimum cuts $S_1 \subset S_2 \cdots \subset S_\ell$.

Step 2: Output the optimal solution $\mathbf{x}^*$ where for $j \in S_k - S_{k-1}, x_j^* = \lambda_{k-1} + 1$.

The complexity of the algorithm is $O(mn \log \frac{n^2}{m} + n \log U)$.

7. Special cases.

7.1. The quadratic CCC problem. Nonlinear and nonquadratic optimization problems with linear constraints were proved impossible to solve in strongly polynomial time in a complexity model of the arithmetic operations, comparisons, and the rounding operation [11]. That negative result, however, is not applicable to the quadratic case, and thus it may be possible to solve constrained quadratic optimization problems in strongly polynomial time. Yet, few quadratic optimization problems are known to be solvable in strongly polynomial time. For instance, it is not known how to solve the minimum quadratic cost network flow problem in strongly polynomial time. For the convex quadratic cost closure problem our result adds to the limited repertoire of quadratic problems solved in strongly polynomial time.

In the quadratic case our algorithm is implemented to run in strongly polynomial time. This is easily achieved since the derivative functions are linear—a case that is shown in [7] to be solved in $O(mn \log \frac{n^2}{m})$. Thus the overall run time of the algorithm is dominated by the complexity of the minimum cut,

$$O\left(mn \log \frac{n^2}{m}\right).$$

7.2. Isotonic regression. The isotonic regression problem is a special case of CCC in which the order is linear and the corresponding graph $G = (V, A)$ is a path from node n to node 1. In other words, the inequalities associated are of the type

$$x_i \le x_{i+1} \quad \text{for all} \quad i = 1, \ldots, n.$$

There are only n possible cuts in such a graph, each with a source set S_i of the form $S_i = \{1, \ldots, i\}$. Each cut is thus $(\{1, \ldots, i\}, \{i+1, \ldots, n\})$. The minimum cut for such graphs is trivially identified in $O(n)$ time by comparing the capacities of the n possible cuts. The capacity of cut $(S_i, \bar{S}_i)$ is computed in $O(1)$ by subtracting from the capacity of $(S_{i-1}, \bar{S}_{i-1})$ the weight of node i, w_i. Indeed, if the weight w_i is positive, then it contributes w_i to the capacity of the cut $(S_{i-1}, \bar{S}_{i-1})$ but not to the capacity of the cut $(S_i, \bar{S}_i)$. If $w_i < 0$, then node i contributes $-w_i$ to the capacity of $(S_i, \bar{S}_i)$ but 0 to the capacity of $(S_{i-1}, \bar{S}_{i-1})$.

Consider the closure graph in which each node has a weight $f_j'(x)$ associated with it for a given value of x. Minimizing the value of the cut is equivalent to minimizing the sum of weights of the sink set (see section 3). Alternatively, the cut is minimized

when the weight of the corresponding source set is maximized, thus seeking an index i to maximize $F_i(x) = \sum_{j=1}^{i} f_j'(x)$. We thus conclude with the following.

LEMMA 7.1. *If* $\sum_{j=1}^{i} f_j'(x) = \max_{k=1,\ldots,n} \sum_{j=1}^{k} f_j'(x)$, *then the minimum cut in the graph* G_x *is* $(S_i, \bar{S}_i)$.

Consider the partial sum functions

$$F_1(x), F_2(x), \ldots, F_n(x),$$

where $F_i(x) = \sum_{j=1}^{i} f_j'(x)$. Recall that the functions $f_j'(x)$ are monotone nondecreasing in x. Denote the roots of the partial sum functions by b_i. Thus $F_i(b_i) = 0$. If the function is negative in the interval $[\ell, u]$, then we let $b_i = u + 1$. If the function is positive throughout the interval, then we let $b_i = \ell - 1$. Let $b_{i_1} = \min_i b_i$. Then for $x \leq b_{i_1}$ the optimal minimum cut is $(\emptyset, V)$. For this cut, the maximum weight source set is empty since all the partial sums of weights are nonpositive. The value of $\lambda_1 = b_{i_1}$ is thus a breakpoint beyond which, for $x > b_{i_1}$, the source set of the minimum cut is $\{1, \ldots, i_1\}$.

As the value of x increases sufficiently so that $\sum_{j=i_1+1}^{i_2} f_j'(x) = F_{i_2}(x) - F_{i_1}(x) \geq 0$, the nodes $\{i_1, \ldots, i_2\}$ join the source set of the minimum cut. In other words, the second breakpoint is the smallest value λ_2 so that there is an index $i_2 > i_1$ such that

$$F_{i_2}(\lambda_2) - F_{i_1}(\lambda_2) \geq 0.$$

The general procedure is as follows:

PROCEDURE ISOTONIC REGRESSION BREAKPOINTS.
$i_0 = 0$, $\lambda_0 = \ell - 1$, $k = 1$
while $i_{k-1} < n$, **do**
Find smallest integer value of λ_k such that for $i_k > i_{k-1}$, $F_{i_k}(\lambda_k) - F_{i_{k-1}}(\lambda_k) \geq 0$.
$k \leftarrow k + 1$
repeat
Output $\lambda_1, \ldots, \lambda_k$.
end

A naive implementation of this algorithm has n iterations with each iteration involving the finding of the roots of $O(n)$ functions. The total complexity is $O(n^2 \log U)$. We can do better with the implementation of the parametric search algorithm that requires the solution of the minimum cut problem for a specific parameter value in $O(n)$. However, each time the procedure calls for the minimum cut, the weights of the nodes must be updated for the new parameter value. This update requires $O(n)$ operation. The additional work of finding the roots of the n functions adds up to a total complexity of $O(n^2 + n \log U)$.

To achieve an even better running time we investigate further the properties of the partial sum functions $F_i(x)$. These functions are obviously monotone nondecreasing as sums of monotone nondecreasing functions. Another important property proved in the next lemma is that each pair of such functions intersects at most once.

LEMMA 7.2. *For* $i < j$ *and functions* F_i *and* F_j, *if for some value of the argument* λ, $F_i(\lambda) < F_j(\lambda)$, *then* $F_i(x) < F_j(x)$ *for any* $x > \lambda$.

Proof. $F_j(x) - F_i(x)$ is a sum of monotone nondecreasing functions $\sum_{k=i+1}^{j} f_k'(x)$. Thus the difference $F_j(x) - F_i(x) > F_j(\lambda) - F_i(\lambda) > 0$ and can increase only as the value of x grows. Thus the two functions do not intersect for any value of $x > \lambda$. □

An immediate corollary of the lemma is that any pair of functions F_i, F_j can intersect at most once.

Consider the upper envelope of the functions F_i represented as an array of functions and breakpoints $(\ell, 0, b_{i_1}, F_{i_1}, b_{i_2}, F_{i_2}, \ldots, b_{i_n}, F_{i_n}, u)$. The functions on the envelope have the property that for all j,

$$F_{i_k}(x) \geq F_j(x), \quad x \in [b_{i_k-1}, b_{i_k}].$$

From the lemma it follows that the upper envelope of the partial sums functions has at most n breakpoints, where the function on the envelope changes. The first breakpoint is b_{i_1}. The next breakpoint occurs for a value of x when some function $F_{i_2}(x) = F_{i_1}(x)$. It is easy to see from Procedure isotonic regression breakpoints that the list of breakpoints of this envelope is precisely the list of the breakpoints that determine the sequence of cuts.

The following *sweep* algorithm may be used to find the upper or upper envelope of a set of functions: Partition arbitrarily the set of functions into two equally sized sets $\mathcal{F}_1$, $\mathcal{F}_2$. Compute recursively the upper envelopes of $\mathcal{F}_1$, $\mathcal{F}_2$. Let E_1, E_2 denote the two resulting upper envelopes. *Sweep* the two upper envelopes E_1, E_2 from left to right and compute the upper envelope of the two upper envelopes. For a detailed description of the above algorithm the reader is referred to [22, pp. 134–136] and [5].

It remains to show how to implement the sweep algorithm for our particular set of functions. Instead of partitioning arbitrarily the set of functions, we choose the partition of $\mathcal{F}_1 = \{1, \ldots, n\}$ to $\{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ and $\mathcal{F}_2 = \{\lfloor \frac{n}{2} + 1, \ldots, n \rfloor\}$. That is, one set contains the lower half-set of indices and the other set contains the upper half-set of indices.

Consider the first breakpoint in E_1 and E_2 (recall that at that point the partial sum values are still 0). If the first breakpoint of E_1 is larger than the first breakpoint of E_2, then the first portion of E_1 is below the first portion of E_2. From the lemma we see that no pairs of functions from the two sets intersect, and the entire envelope E_1 lies below the envelope E_2. Thus the merged envelope is E_2.

If, on the other hand, the first breakpoint of E_1 is smaller than the first breakpoint of E_2, then there could be a point where a function from $\mathcal{F}_2$ crosses a function from $\mathcal{F}_1$. We consider the array of breakpoints of the envelope E_1 for the last breakpoint, where it is still above E_2. Similarly, we search the array of breakpoints of the envelope E_2 for the last breakpoint, where it is still below E_1. Since there are $O(n)$ breakpoints per envelope, the search for that breakpoint is done by binary search in $O(\log n)$ steps. The intersection point is then to be determined between this breakpoint and the next one on each envelope. Finding the intersection of $F_i(x)$ and $F_j(x)$ takes at most $O(\log U)$ steps.

Thus the merger of two envelopes of functions is executed in $O(\log n + \log U)$. Since there are at most n mergers in the procedure, the total running time is $O(n \log n + n \log U)$.

Once all the upper envelopes have been identified we have the implied source sets of the associated cuts:

$$\{1, \ldots, i_1\}, \ \{1, \ldots, i_2\}, \ldots, \{1, \ldots, i_q\}.$$

If $i \in \{i_{k-1}, \ldots, i_k\}$, then $x_i^* \in (b_{i_{k-1}}, b_{i_k}]$. It remains to apply Lemma 6.1 in order to determine an optimal solution:

$$x_i^* = b_{i_{k-1}} + 1.$$

Thus the total complexity of the algorithm for the isotonic regression problem is $O(n(\log U + \log n))$. In the quadratic case this leads to a complexity of $O(n \log n)$.

8. The continuous CCC problem. When solving the problem in continuous variables, one has to determine how to output the solution. For instance, the minimum of a cubic function can be irrational even if all coefficients are integers. To fully provide the output would then require infinite complexity. To that end we employ the ϵ-accurate complexity model introduced in [10]. According to this model a solution $\mathbf{x}^{(\epsilon)}$ is specified as an integer multiple of ϵ; i.e., it lies on the so-called ϵ-grid. The solution is such that there is an optimal vector $\mathbf{x}^*$ so that

$$||\mathbf{x}^{(\epsilon)} - \mathbf{x}^*||_\infty < \epsilon.$$

The continuous problem can be solved using the same algorithm used for the integer case. The only modification required is in the parametric analysis procedure where the choice of λ^* is such that a median point in the interval (λ_1, λ_2) lies on the ϵ-grid. This is done in additional $O(n \log(U/\epsilon)$ time. The complexity of the algorithm is thus the complexity of finding the roots of the n functions plus the complexity of a minimum cut, $O(mn \log \frac{n^2}{m} + n \log(U/\epsilon))$.

9. Conclusions and extensions. The results here have been extended to a problem that is more general than the CCC problem. The problem is the *convex s-excess problem* which generalizes the s-excess problem discussed in [13]. The problem is formulated as follows:

$$\begin{array}{lll} \text{(Convex s-excess)} & \text{Min} & \sum_{j\in V} f_j(x_j) + \sum e_{ij} z_{ij} \\ \text{subject to} & & x_i - x_j \le z_{ij} \quad \text{for } (i,j) \in A, \\ & & u_j \ge x_j \ge \ell_j, \quad j = 1, \ldots, n, \\ & & z_{ij} \ge 0, \qquad (i,j) \in A. \end{array}$$

This problem is solved with precisely the same complexity as the minimum closure problem. To that end, we proved a generalization of the threshold theorem reported in [15].

There are applications of the convex s-excess problem in the areas of image segmentation and Markov random fields. The problem is of further interest because of its relationship to the minimum cost network flow problem.

Notice that the terms associated with the variables z_{ij} are linear. This is significant because the dual of the minimum cost network flow (MCNF) problem is

$$\begin{array}{lll} \text{(Dual MCNF)} & \text{Min} & \sum_{j\in V} b_j x_j + \sum e_{ij} z_{ij}, \\ \text{subject to} & & x_i - x_j \le c_{ij} + z_{ij} \quad \text{for } (i,j) \in A, \\ & & u_j \ge x_j \ge \ell_j, \quad j = 1, \ldots, n, \\ & & z_{ij} \ge 0, \qquad (i,j) \in A. \end{array}$$

That is, the right-hand sides of the constraints have a constant term in addition to the term z_{ij}. Thus, if one can solve the convex s-excess problem with convex function term $\sum e_{ij}(z_{ij})$, then it would have been possible to solve also the dual of the MCNF in the same running time for a single application of maximum flow or minimum cut.

Notes added in proof. (1) The parametric algorithm was shown in [7] to work in strongly polynomial time for linear capacity functions. Hochbaum and Hong [*About strongly polynomial time algorithms for quadratic optimization over submodular constraints,* Math. Programming, 69 (1995), pp. 269–309] showed that the same run

time applies when the monotone capacity functions are piecewise linear. The number of pieces in the piecewise linear functions N adds $O(Nn)$ to the complexity of the parametric minimum cut procedure. Since for CCC the capacity functions are derivatives of convex functions it follows that for convex functions that are piecewise linear or piecewise quadratic the run time remains strongly polynomial.

(2) In [13] Hochbaum shows that the pseudoflow algorithm solves the maximum flow minimum cut algorithm for tree graphs in $O(n)$ steps. In tree graphs the set of arcs other than those adjacent to source and sink form an (undirected) acyclic graph. Thus when the partial order graph is a tree the CCC is solved in $O(n \log U)$ using the procedure of [13] with the binary search algorithm described in section 6. This is an alternative algorithm to solve the isotonic regression problem where the linear order graph is a path and thus a tree.

Acknowledgments. The first author thanks Sariel Har-Peled for his input on the state of the art of manipulating lower and upper envelopes of functions and for pointing out references [5], [21], and [22]. The first author also is grateful to Günter Rote for pointing out a simplification in the procedure.

REFERENCES

[1] R. K. Ahuja and J. B. Orlin, *A fast scaling algorithm for minimizing separable convex functions subject to chain constraints*, Oper. Res., 49 (2001), pp. 784–789.
[2] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin, *A Cut Based Algorithm for the Convex Dual of the Minimum Cost Network Flow Problem*, manuscript.
[3] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin, *Solving the convex cost integer dual network flow problem*, Management Sci., to appear.
[4] R. E. Barlow, D. J. Bartholomew, J. M. Bremer, and H. D. Brunk, *Statistical Inference Under Order Restrictions*, Wiley, New York, 1972.
[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.
[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1989.
[7] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, *A fast parametric maximum flow algorithm and applications*, SIAM J. Comput., 18 (1989), pp. 30–55.
[8] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, J. Assoc. Comput. Mach., 35 (1988), pp. 921–940.
[9] D. S. Hochbaum and J. Naor, *Simple and fast algorithms for linear and integer programs with two variables per inequality*, SIAM J. Comput., 23 (1994), pp. 1179–1192.
[10] D. S. Hochbaum and J. G. Shanthikumar, *Convex separable optimization is not much harder than linear optimization*, J. Assoc. Comput. Mach., 37 (1990), pp. 843–862.
[11] D. S. Hochbaum, *Lower and upper bounds for allocation problems*, Math. Oper. Res., 19 (1994), pp. 390–409.
[12] D. S. Hochbaum, *Solving integer programs over monotone inequalities in three variables: A framework for half integrality and good approximations,* European J. Oper. Res., 140 (2002), pp. 291–321.
[13] D. S. Hochbaum, *The Pseudoflow Algorithm for the Maximum Flow Problem*, manuscript.
[14] D. S. Hochbaum, *Efficient algorithms for the inverse spanning tree problem*, Oper. Res., to appear.
[15] D. S. Hochbaum, *An efficient algorithm for image segmentation, Markov random fields and related problems*, J. Assoc. Comput. Mach., 48 (2001), pp. 686–701.
[16] W. L. Maxwell and J. A. Muckstadt, *Establishing consistent and realistic reorder intervals in production/distribution systems*, Oper. Res., 33 (1985), pp. 1316–1341.
[17] J. C. Picard, *Maximal closure of a graph and applications to combinatorial problems*, Management Sci., 22 (1976), pp. 1268–1272.
[18] J.-C. Picard and M. Queyranne, *Selected applications of minimum cuts in networks*, INFOR—Canad. J. Oper. Res. Inform. Process., 20 (1982), pp. 394–422.
[19] J. C. Picard and M. Queyranne, *Integer Minimization of a Separable Convex Function Subject to Variable Upper Bound Constraints*, manuscript.

[20] R. Roundy, *A 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems*, Management Sci., 31 (1985), pp. 1416–1430.
[21] M. I. Shamos and D. Hoey, *Geometric intersection problems*, in Proc. 17th Ann. Symp. on Foundations of Computer Science, IEEE, Long Beach, CA, 1976, pp. 208–215.
[22] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.
[23] P. T. Sokkalingam, R. Ahuja, and J. B. Orlin, *Solving inverse spanning tree problems through network flow techniques*, Oper. Res., 47 (1999), pp. 291–298.
[24] A. F. Veinott, Jr., *Least d-majorized network flows with inventory and statistical applications*, Management Sci., 17 (1971), pp. 547–567.