Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Incremental Network Optimization: Theory and Algorithms

Onur Şeref, Ravindra K. Ahuja, James B. Orlin,

To cite this article:
Onur Şeref, Ravindra K. Ahuja, James B. Orlin, (2009) Incremental Network Optimization: Theory and Algorithms. Operations Research 57(3):586-594. http://dx.doi.org/10.1287/opre.1080.0607

OPERATIONS RESEARCH
Vol. 57, No. 3, May–June 2009, pp. 586–594
ISSN 0030-364X | EISSN 1526-5463 | 09 | 5703 | 0586

informs

DOI 10.1287/opre.1080.0607

Incremental Network Optimization: Theory and Algorithms

Onur Şeref
Department of Business Information Technology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061, seref@vt.edu

Ravindra K. Ahuja
Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida 32611, ahuja@ufl.edu

James B. Orlin
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, jorlin@mit.edu

In an incremental optimization problem, we are given a feasible solution x^0 of an optimization problem P, and we want to make an incremental change in x^0 that will result in the greatest improvement in the objective function. In this paper, we study the incremental optimization versions of six well-known network problems. We present a strongly polynomial algorithm for the incremental minimum spanning tree problem. We show that the incremental minimum cost flow problem and the incremental maximum flow problem can be solved in polynomial time using Lagrangian relaxation. We consider two versions of the incremental minimum shortest path problem, where increments are measured via arc inclusions and arc exclusions. We present a strongly polynomial time solution for the arc inclusion version and show that the arc exclusion version is NP-complete. We show that the incremental minimum cut problem is NP-complete and that the incremental minimum assignment problem reduces to the minimum exact matching problem, for which a randomized polynomial time algorithm is known.

Subject classifications: theory; distance algorithms; flow algorithms.
Area of review: Optimization.
History: Received March 2007; accepted January 2008. Published online in *Articles in Advance* February 9, 2009.

1. Introduction

We start with a general definition of an incremental optimization problem. Let P be an optimization problem with a starting feasible solution x^0, and let $\mathbf{B}$ be the set of all feasible solutions. For a new feasible solution x, the increment from x^0 to x is the amount of change given by a function $f(x, x^0)$: $\mathbf{B} \times \mathbf{B} \to \mathbf{R}$, which we refer to as the *incremental function*. Suppose that k is a given bound on the total change permitted. We call x an *incremental solution* if it satisfies the inequality $f(x, x^0) \leqslant k$. In an incremental optimization problem, we seek to find an incremental solution x^* that results in the maximum improvement in the objective function value.

Our work on incremental optimization has been motivated by the practice-oriented research of one of the authors on the *railroad blocking problem* (Ahuja et al. 2007). The railroad blocking problem is an important planning problem for freight railroads and determines how to aggregate a large number of shipments into *blocks* of shipments as they travel from origins to destinations. The blocking plan for a railroad dictates a railroad's train schedule, and train schedule, in turn, dictates the flow of three important railroad assets: crews, locomotives, and railcars. The optimal blocking plan is a function of the origin-destination (OD) traffic, and, as OD traffic changes over time, the blocking plan must be reoptimized to obtain a new blocking plan. If the new blocking plan is considerably different from the previous blocking plan, then the train schedule needed to carry the new set of blocks may be considerably different from the previous train schedule. A different train schedule will result in different locomotive and crew assignments. Needless to say, railroads are averse to changing blocking plans dramatically from month to month and instead prefer to change the blocking plan incrementally. They want to specify their current blocking plan as the starting plan, the degree of change allowed, and would like to obtain an "incrementally optimal" blocking plan that would differ from the given plan by no more than the specified change. This is an example of incremental optimization. Similar incremental optimization problems can be defined in other railroad planning contexts: train scheduling, locomotive planning, and crew scheduling (Ahuja et al. 2006; Vaidyanathan et al. 2007, 2008). Incremental optimization is essential to use modeling results in practice.

Although the term of "incremental optimization" is new, this concept has been around in various forms. Within the area of optimization and heuristics, local search techniques modify solutions by seeking out improved solutions within a neighborhood, and typically neighborhood solutions are very similar to the original solutions (Aarts and

Lenstra 1997). Typically, the modifications from iteration to iteration are small; however, there can be more significant changes per iteration in very large-scale neighborhood search (Ahuja et al. 2002) or in genetic algorithms (Holland 1975). In addition, iterative approaches with modest changes in solution are standard in a variety of optimization approaches including many nonlinear programming approaches, linear programming approaches, value improvement approaches in dynamic programming, and elsewhere.

Incremental optimization can also be of great value for a manager to learn about models. For example, Little (1970) points out that "most models are incomplete" and managers typically learn about models by changing them slightly and seeing the impact on the solution. We suggest that, for a manager to learn about a model, it helps if a small change in the model leads to a small change in the solution. A closely related idea is "iterative algorithms" or "human-machine scheduling" in which the user guides a machine to improved solutions (Dos Santos and Bariff 1988, Higgins 1995). For a human to recognize one solution as being superior to another, it greatly helps if the two solutions are largely the same so that the human can focus on a small number of differences; thus, there is a strong advantage for incremental optimization from step to step. Incremental optimization is also a natural way of letting the manager understand the model better. Incremental optimization can also be a valuable classroom tool, where one improves a model one step at a time while developing confidence along the way that the model is doing what it should (Regan 2006).

Incremental optimization is also related to "rolling-horizon schedules," where one schedules the next p periods and then fixes the next period only (Wagner 1977). Incremental optimization is also of value in situations in which there are multiple optimizers who need to learn information within the process. For example, Adomavicius and Gupta (2005) discuss how iterative rounds in which prices change small amounts in each round can lead to more effective combinatorial auctions. Incremental optimization is also closely linked to the concept of "continuous improvement" and the related concept of the learning curve (Zangwill and Kantor 1998, Li and Rajagopalan 1998). If one makes a large change in a process, it is much more difficult to learn what features of the change led to the improvements. On the other hand, continual incremental changes lead to a greater degree of learning.

In this paper, we study incremental optimization on a number of network flow problems. This paper is organized as follows. In §2, we study the incremental minimum spanning tree, incremental minimum cost flow, and incremental maximum flow problems. We introduce a Lagrangian relaxation approach that converts these problems into parametric problems without the incremental constraint. Then, by exploiting the special structure of the parametric problems and performing binary search on a single parameter, we develop efficient algorithms to solve these problems.

In §3, we study three other incremental network problems: the incremental shortest path problem, the incremental minimum cut problem, and the incremental minimum assignment problem. We consider the incremental shortest path problem with two different versions with respect to the increments: arc inclusion and arc exclusion; we show that the first version can be solved in polynomial time, whereas the second version is an NP-complete problem. We show that the incremental minimum cut problem is NP-complete, and the incremental minimum assignment problem reduces to the minimum exact matching problem, for which a randomized polynomial time algorithm is known.

2. Lagrangian Relaxation and Binary Search-Based Algorithms

In this section, we introduce an approach that uses a combination of Lagrangian relaxation and binary search to solve incremental network problems. The Lagrangian relaxation is performed by carrying the incremental constraint to the objective function and converting the incremental problem to a parametric network problem. Then, from the special structure of the latter problem, we use a binary search algorithm to find the parameter for which the solution to the parametric problem is also the solution for the incremental problem.

The Lagrangian relaxation of an optimization problem $(P^{\leqslant}) = \min\{cx\colon\ Ax \leqslant b, x \in X\}$ is the optimization problem $L(\lambda) = \min\{cx + \lambda(Ax - b), x \in X\}$, where λ is a vector of parameters. For optimization problems in which the inequality constraints are relaxed, the following theorem gives conditions such that the solution to the Lagrangian relaxation is also the solution to the original problem.

THEOREM 2.1. *Let $P^{\leqslant}$ be an optimization problem to which we apply Lagrangian relaxation by relaxing the inequalities $Ax \leqslant b$. Suppose that x^* is a solution to the relaxed problem for some Lagrangian multiplier vector $\lambda \geqslant 0$. If x^* is feasible in $P^{\leqslant}$ and satisfies the complementary slackness condition $\lambda(Ax^* - b) = 0$, then it is also an optimal solution to $P^{\leqslant}$.*

Theorem 2.1 is well known. See, for example, Ahuja et al. (1993). In the three incremental optimization problems that we introduce in this section, the incremental constraint is a single constraint. Therefore, the "vector" of Lagrangian multipliers is a single parameter λ on which we can use binary search to find a solution that satisfies the conditions given in Theorem 2.1. Next, we introduce the incremental minimum spanning tree problem, the incremental minimum cost flow problem, and the incremental maximum flow problem.

2.1. Incremental Minimum Spanning Tree Problem

In an incremental minimum spanning tree problem, we are given an initial spanning tree T^0 in an undirected graph $G = (N, A)$. We want to find a spanning tree T^* of minimum

possible cost that has at most k arcs different from T^0. In this section, we show that the incremental minimum spanning tree problem can be solved with a strongly polynomial time algorithm in which we relax the original problem into a parametric minimum spanning tree problem and perform a binary search on the parameter.

Throughout this section, we let T denote both a spanning tree and the set of arcs in it. We denote the cost of an arc $e \in G$ as $c(e)$ and the cost of a tree as $c(T) = \sum_{e \in T} c(e)$. We define the increment function as $f(T, T^0) = |T \backslash T^0|$, the number of arcs in T and not in T^0. An incremental tree is a tree T with $f(T, T^0) \leqslant k$. Now, we can define the problem as follows.

Definition 2.2 (Incremental Minimum Spanning Tree Problem). Let $G = (N, A)$ be an undirected graph and let T^0 be a spanning tree in G. Find a tree T^k such that $c(T^k) = \min\{c(T)\colon f(T, T^0) \leqslant k\}$, where $f(T, T^0) = |T \backslash T^0|$.

Before we introduce our algorithm, we present a lemma showing that for any minimum spanning tree T^*, there is an incremental minimum spanning tree T^k in the subgraph $G^* = (N, A^*)$, where $A^* = T^0 \cup T^*$. Assuming that $|N| = n$, this theorem implies that the number of arcs needed to be considered by an algorithm is $|A^*| = 2n - 2$ in the worst case.

When we remove an arc e from a spanning tree T in G, we include a partition of the nodes into subsets N_1 and N_2, which are nodes of the two subtrees. This partition of nodes is a *cut*. We let $Q(T, e)$ denote the set of arcs in the cut, which is the set of all arcs $e \in A$ with one end in N_1 and the other end in N_2.

Lemma 2.3. *For any minimum spanning tree T^* in G, there exists a minimum cost incremental spanning tree T^k in the subgraph $G^* = (N, A^*)$, where $A^* = T^* \cup T^0$.*

Among all the solutions to the incremental minimum spanning tree problem, let T^k be the one with a maximum number of arcs in A^*. If $T^k \subseteq A^*$, then the proof is complete. So, suppose that $e \in T^k \backslash A^*$. By the cut property of the optimum spanning tree (see, for example, Ahuja et al. 1993), there is a minimum cut arc e in $Q(T^k, e)$, which is in T^*. Thus, $T^k + e' - e$ is also an incremental tree with cost at most $c(T^k)$, violating that T^k has the maximum number of arcs in A.

2.1.1. IP Formulation and Lagrangian Relaxation. We give a mathematical programming formulation of the incremental minimum spanning tree problem below and show that the Lagrangian relaxation of this formulation is equivalent to a parametric minimum cost spanning tree problem. Then, we present the conditions for which the solution to the parametric minimum cost spanning tree problem is also a solution to the incremental minimum spanning tree problem.

Let T^* be a minimum spanning tree in G. From Lemma 2.3, we can restrict our attention to the subgraph $G^* = (N, A^*)$, where $A^* = T^* \cup T^0$. Let P denote the following problem:

$$\min \quad cx$$

$$\text{subject to} \quad \sum_{e_i \in \{T^* \backslash T^0\}} x_i \leqslant k, \tag{1a}$$

$$x \in X, \tag{1b}$$

$$x_i = 0 \quad \text{for } i \notin A^*. \tag{1c}$$

In this formulation, $\{T^* \backslash T^0\}$ is the set of nontree arcs and $x = \{x_i\}$ is a vector of binary decision variables, where $x_i = 1$ indicates that arc e_i is in the solution and $x_i = 0$ indicates that arc e_i is not in the solution. The vector c denotes the arc costs $c(e_i)$, and the set X denotes the set of solution vectors of all possible spanning trees in G^*. Constraint (1c) enforces that at most k arcs not in T^0 can be in an incremental spanning tree solution.

Let n be the number of nodes in G^*. Note that for each arc included from the set $\{T^* \backslash T^0\}$ in the solution, we delete an arc from T^0; therefore, constraint (1c) can be replaced by the following constraint, which states that there should be at least $n - k - 1$ arcs that are in T^0 in the incremental spanning tree solution:

$$\sum_{e_i \in T^0} x_i \geqslant (n - k - 1). \tag{2}$$

We associate a Lagrangian multiplier with constraint (2) and relax it to get the following Lagrangian subproblem:

$$L(\lambda) = \min\left\{cx - \lambda \sum_{e_i \in T^0} x_i + \lambda(n - k - 1)\colon x \in X\right\} \tag{3}$$

$$= \min\left\{\sum_{e_i \in T^0} (c(e_i) - \lambda)x_i - \sum_{e_i \in \{T^* \backslash T^0\}} c(e_i)x_i\colon x \in X\right\}. \tag{4}$$

So, the solution to $L(\lambda)$ is a minimum cost spanning tree problem for each λ, and we denote it by $T(\lambda)$.

We now perturb the arc costs by replacing $c(e_i)$ with $c'(e_i) = c(e_i) + \phi(i)$, where $\phi(i) = (mi^2 + i)/(m + 1)^3$. These perturbed values satisfy the following easily verified properties (Dimitromanolakis 2002):

(P1) $c(e_i) < c'(e_i) < c(e_i) + 1$.

(P2) For i, j, k, and l with $i \neq j$ and $i \neq k$, $c'(e_i) - c'(e_j) \neq c'(e_k) - c'(e_l)$.

Theorem 2.4. *Let G' be the perturbed graph with arc costs $c(i) + \phi(i)$, and let T be a minimum spanning tree in G. Then, T is also a minimum spanning tree in the original graph G.*

Let T^* be the minimum spanning tree for G' as obtained by the greedy algorithm for G'. Then, T^* is also obtained by the greedy algorithm for G because of property (P1)

assuming that arcs are listed in terms of increasing costs for c'.

We assume that the arc costs $c(e_i)$ are replaced by the perturbed arc costs for the remainder of §2.1. The following theorem states the conditions on the solution to the parametric minimum spanning tree for which the solution is also an incremental minimum spanning tree. Let $T(\lambda)$ be a solution to the parametric minimum spanning tree problem.

THEOREM 2.5. *Suppose that the cost function satisfies properties* (P1) *and* (P2). *A spanning tree T is a* (*unique*) *minimum cost incremental spanning tree if and only if there is a value $\lambda \geqslant 0$ such that T is optimal for $L(\lambda)$ and*

(i) $f(T, T^0) \leqslant k$ *and* $\lambda = 0$ *or*

(ii) $f(T, T^0) = k$ *and* $\lambda \neq 0$.

The uniqueness of the minimum spanning tree follows from property (P2), which implies $c(e_i) \neq c(e_j)$ for any $i \neq j$. The "*if*" part follows directly from Theorem 2.1. For the "*only if*" part, first note that a minimum spanning tree T^* is the solution to $L(0)$. If $f(T^*, T^0) \leqslant k$, then T^* is clearly an incremental minimum spanning tree. Suppose that $f(T^*, T^0) = t > k$. If we increase λ, then the arcs in T^0 will have a lower cost. At some λ_1, one of the nontree arcs $e_0 \in \{T^0 \backslash T(\lambda_0)\}$ will have the same cost as a tree arc $e_0^* \in \{T(\lambda_0) \backslash T^0\}$; i.e., $c(e_0) - \lambda_1 = c(e_0^*)$. Note that, because of the perturbation in the arc costs, the arc pair e_0^* and e_0 is unique. Then, we can replace e_0^* with e_0 to get $T(\lambda_1)$ such that $f(T(\lambda_1), T^0) = t - 1$. If we continue increasing λ, we generate a unique series of trees $T(\lambda_1), T(\lambda_2), \ldots, T(\lambda_{(t-k)}) = T^0$ with $f(T(\lambda_i), T^0) = t - i$. This implies that $f(T(\lambda), T^0)$ is a monotonically decreasing function of λ and $f(T(\lambda^*), T^0) = k$ at some $\lambda^* = \lambda_{(t-k)} = c(e_{(t-k)}) - c(e^*_{(t-k)})$ for some $e = e_{(t-k)} \in \{T^0 \backslash T(0)\}$ and $e^* = e^*_{(t-k)} \in \{T(0) \backslash T^0\}$.

The proof of Theorem 2.5 immediately leads to a strongly polynomial time algorithm because one can find the value λ_j in polynomial time for each j. A standard implementation runs in $O(n^2)$ time for each λ_j, for a total running time of $O(n^2 k)$. This can be improved using sophisticated data structures such as dynamic trees, which improve the running time to $O(nk \log n)$. However, one can do even better by doing binary search in a clever manner, reducing the total running time to $O(n \log n)$. We describe such a binary search in the next subsection.

2.1.2. A Faster Strongly Polynomial Algorithm. We want to develop a strongly polynomial algorithm for finding a minimum cost incremental spanning tree. First note that, from the proof of Theorem 2.5, we know that λ^* is equal to the difference of the cost of two arcs. Given that A^* has at most $2n - 2$ arcs, it means that $1/4n^2 < \lambda^* \leqslant C$. Because $f(T(\lambda), T^0)$ is monotonically decreasing in λ, we can find λ^* using binary search by solving $O(\log nC)$ parametric minimum spanning tree problems. However, we want to solve $O(\log n)$ minimum spanning tree problems in the worst case until we find λ^*. We want to find the optimal parameter λ^* for which $f(T(\lambda^*), T^0) = k$. To simplify the algorithm's presentation, we restrict attention to the case that T^* is not a minimum cost incremental spanning tree.

Following the proof of Theorem 2.5, we can reduce the search for λ^* in a matrix D whose entries are the differences between the arc costs of the sets $\{T^0 \backslash T^*\}$ and $\{T^* \backslash T^0\}$. First, remember that the arc costs are slightly perturbed such that each entry in D is distinct. We construct D as follows: Let L be a list of the arcs $e \in \{T^0 \backslash T^*\}$ in increasing order of their cost, and let L^* be a list of arcs $e^* \in \{T^* \backslash T^0\}$ in decreasing order of their cost. Let the matrix D be such that $d(i, j) = c(e_i) - c(e_j^*)$, where $e_i \in L$ and $e_j^* \in L^*$. We do not create D explicitly because it would result in $\Omega(n^2)$ time. Rather, we show how to carry out approximate binary search while needing only to evaluate $O(n \log n)$ distinct entries in D. Because $f(T(\lambda), T^0)$ is monotonically decreasing with λ and $d(i, j)$ is monotonically increasing with both i and j, we can perform an approximate binary search on D using our algorithm (see Algorithm 1). We will show how to implement certain steps efficiently after describing the algorithm.

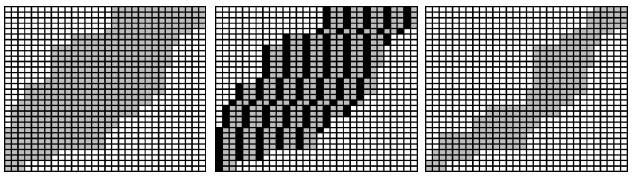
ALGORITHM 1. Procedure that finds the incremental MST.

procedure INCREMENTALMST(U, L)
  for $i = 1$ **to** n **do**
    $MinIndex(i) = \min\{j\colon d(i, j) \geqslant L\}$;
    $MaxIndex(i) = \max\{j\colon d(i, j) \leqslant U\}$;
    if $MinIndex(i) \leqslant MaxIndex(i)$ **then**
      $Count(i) = MaxIndex(i) - MinIndex(i) + 1$;
    else
      $Count(i) = 0$;
    endif
  end for
  $TotalCount = \sum_{i=1}^{n} Count(i)$;
  if $TotalCount \leqslant 12n$ **then**
    $F = \{(i, j)\colon L \leqslant d(i, j) \leqslant U\}$;
    return $BinarySearch(F)$;
  end if
  $K = \lfloor TotalCount/6n \rfloor$;
  for $i = 1$ **to** n **do**
    $H(i) = \{(i, j)\colon L \leqslant d(i, j) \leqslant U,$
      $j = MinIndex(i) + rK - 1,\ r \in \mathbf{Z}^+\}$;
  end for
  $H = \bigcup_{i=1}^{n} H(i)$;
  $\lambda = \text{median}\{d(i, j)\colon (i, j) \in H\}$;
  if $f(T(\lambda), T^0) = k$ **then return** $T(\lambda)$;
  else if $f(T(\lambda), T^0) > k$ **then**
  $IncrementalMST(\lambda, U)$;
  else if $f(T(\lambda), T^0) < k$ **then**
  $IncrementalMST(L, \lambda)$;
  end if
end procedure

The following is a summary of how the algorithm works:

1. Let L be a lower bound on λ^* and U be an upper bound. Find the set of *feasible* pairs (i, j); i.e., $F = \{(i, j) : L \leqslant d(i, j) \leqslant U\}$;

Figure 1. Matrix D with arc differences $c(e_i) - c(e_j)$.



Notes. (Left) Feasible pairs that are between L and U. (Middle) Selection of set H with equidistant feasible entries. (Right) The new feasible region for $L' = median(H)$ and $U' = U$.

2. If $|F| \leqslant 12n$, then enumerate and sort all feasible pairs and do a binary search on the sorted pairs to find the pair with the value λ^* such that $f(T(\lambda^*), T^0) = k$; return $T(\lambda^*)$; STOP;
Else continue;

3. If $|F| > 12n$, then select a subset of H of $O(n)$ pairs that are equally spaced in each row and find the median pair of H; let $\bar{\lambda}$ denote its value.

4. If $f(T(\bar{\lambda}), T^0) = k$, then return $T(\bar{\lambda})$; STOP;
Else continue;

5. If $f(T(\bar{\lambda}), T^0) < k$, then $U = \bar{\lambda}$; else if $f(T(\bar{\lambda}), T^0) > k$, then $L = \bar{\lambda}$; Go to 1.

The key to our algorithm is the way we choose a subset H of $O(n)$ pairs and how it leads to eliminating a constant fraction of the feasible pairs each time we perform the above steps from 1 to 5 until we have $12n$ or fewer pairs. Steps 1, 3, and 5 are shown in Figure 1. Theorem 2.6 proves our algorithm to be strongly polynomial.

THEOREM 2.6. *The incremental minimum spanning tree problem can be solved in $O(m\alpha(m, n) + (n \log n)\alpha(n, n))$ time, where α is the inverse Ackermann function.*

First, we find a minimum spanning tree T^* in G and check whether $f(T^*, T^0) \leqslant k$. Chazelle (2000) shows that, in a graph with n nodes and m arcs, T^* can be found in $O(m\alpha(m, n))$ time, where α is the inverse Ackerman function, an extremely slowly growing function. Now we can reduce the graph to $G^* = (N, A^*)$, where $A^* = T^* \cup T^0$.

If $f(T^*, T^0) > k$, then we call Algorithm 1. $MaxIndex(i)$ ($MinIndex(i)$) is the feasible pair with the maximum (minimum) index in row i of D. The pairs $d(i, j)$ are monotonically increasing with i and j; therefore, $MaxIndex(i) \geqslant MaxIndex(i + 1)$. Knowing $MaxIndex(i)$, we can search for $MaxIndex(i + 1)$ starting at $j = MaxIndex(i)$. A similar argument is valid with $MinIndex(i) \geqslant MinIndex(i + 1)$; therefore, starting with $d(1, 1)$, the for-loop between lines 2 and 10 takes $O(n)$ time. Moreover, the number of feasible pairs can be calculated by taking the sum of $MaxIndex(i) - MinIndex(i) + 1$ over i in $O(n)$ time. We consider two cases depending on the number of the feasible pairs:

1. If the number of feasible pairs is $12n$ or less, the algorithm calls the *BinarySearch* procedure, which enumerates and sorts the feasible pairs in $O(n \log n)$ time. Using binary search on the sorted list, in the worst case, it solves $O(\log n)$ minimum spanning tree problems $T(\lambda)$. Because G^* has at most $2n - 2$ arcs, each minimum spanning tree problem takes $O(n\alpha(n, n))$ time; thus, *BinarySearch* runs in $O((n \log n)\alpha(n, n))$ time.

2. If the number of feasible pairs is greater than $12n$, then the algorithm creates a subset of feasible pairs H. For each row i, starting from the first feasible pair, $MinIndex(i)$, it includes every Kth feasible pair, where $K = \lfloor TotalCount/6n \rfloor$. Note that $6n \leqslant |H| \leqslant 9n$, and knowing $MinIndex(i)$ and $MaxIndex(i)$, it takes $O(n)$ time to create H. The median $\bar{\lambda}$ of H can be found in $O(n)$ time, and the minimum spanning tree $T(\bar{\lambda})$ can be found in $O(n\alpha(n, n))$ time. If $f(T(\bar{\lambda}), T^0) = k$, then we return $T(\bar{\lambda})$ as an incremental minimum spanning tree. If $f(T(\bar{\lambda}), T^0) \neq k$, then there are two cases. Assume that λ^* is the feasible pair we want to find.

(a) *Case* 1. $f(T(\lambda), T^0) > k$ and $\lambda < \lambda^*$. Then, there are at least $3n$ pairs in H with $d(i, j) \leqslant \lambda$. Note that the number of feasible pairs with $d(i, j) \leqslant \lambda$ is at least $3nK \geqslant TotalCount/2 - 3n \geqslant TotalCount/4$. In this case, we can eliminate at least 1/4 of the feasible pairs.

(b) *Case* 2. $f(T(\lambda), T^0) < k$ and $\lambda > \lambda^*$. Then, there are at least $3n$ pairs in H with $d(i, j) \geqslant \lambda$. In this case, the number of feasible pairs with $d(i, j) \geqslant \lambda$ is at least $2nK \geqslant TotalCount/3 - 3n \geqslant TotalCount/6$, and we can eliminate at least 1/6 of the feasible pairs.

Each time Case 2 occurs at least 1/6 of the feasible pairs are eliminated, and, in the worst case, we solve $O(\log n)$ minimum spanning tree problems, which takes $O((n \log n)\alpha(n, n))$ time, until Case 1 occurs. Thus, our Algorithm 1 returns an incremental minimum spanning tree in $O((n \log n)\alpha(n, n))$ time. Together with the time to find the initial minimum spanning tree T^*, we can find an incremental minimum spanning tree in $O(m\alpha(m, n) + (n \log n)\alpha(n, n))$ time.

2.2. Incremental Network Flow Problems

In this section, we study two incremental network flow problems: the incremental minimum cost flow problem and the incremental maximum flow problem. First, we convert the incremental minimum cost flow problem into an incremental minimum cost circulation problem, and we show that this incremental problem is equivalent to a minimum cost circulation problem with one additional linear constraint, which is known to have a polynomial time solution. Finally, we show that the incremental maximum flow problem can also be reduced to an incremental minimum cost circulation problem with arc costs $|c_{ij}| \leqslant 1$.

2.2.1. Incremental Minimum Cost Flow Problem. In an incremental minimum cost flow problem, we are given an initial flow x^0 on a capacitated network G and we want to find a flow x^* that differs from the initial flow at most by some allowed incremental amount k while minimizing the cost of the new flow. We do not require that flows be integer valued. Let $G = (N, A)$ be a capacitated network with arc capacities $u = \{u_{ij}\}$ and arc costs $c = \{c_{ij}\}$ for

all $(i, j) \in A$. Each node in the network has a supply or a demand of $d(i)$ depending on whether $d(i) > 0$ or $d(i) < 0$, respectively. The supply is transferred through the arcs to satisfy the demand by the *flow* $x = \{x_{ij}\}$, where x is the vector of the flow values on the arcs. The minimum cost flow problem can be stated as follows:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5a}$$

$$\text{subject to} \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A(x^0)} x_{ji} = d(i) \quad \text{for all } i \in N, \tag{5b}$$

$$0 \leqslant x_{ij} \leqslant u_{ij} \quad \text{for all } (i,j) \in A. \tag{5c}$$

Equation (5b) is referred to as the mass-balance constraints, and any flow that satisfies these constraints is a feasible flow. We represent the set of feasible flows as X. Let x^0 be an initial feasible flow. We define the increment function for a feasible flow x as $f(x, x^0) = \sum_{(i,j)\in A} |x_{ij} - x^0_{ij}|$, which is the sum of the absolute changes in the arc flows between x and x^0. We define the incremental minimum cost flow problem as follows.

Definition 2.7 (Incremental Minimum Cost Flow). Let $G = (N, A)$ be a capacitated graph with arc capacities $u = \{u_{ij}\}$ and arc costs $c = \{c_{ij}\}$. Let x^0 be an initial feasible flow. Find a feasible flow x^* such that $cx^* = \min\{cx\colon\ f(x, x^0) \leqslant k, x \in X\}$, where $f(x, x^0) = \sum_{(i,j)\in A} |x_{ij} - x^0_{ij}|$.

It can be shown for any feasible solution in G that there is a feasible solution in the residual graph $G(x^0)$ and vice versa, such that the values of the cost of both solutions are equal. Theorem 2.8 summarizes this equivalence.

Theorem 2.8. *A flow x is a feasible flow in graph G if and only if its corresponding flow x', defined by $x'_{ij} - x'_{ji} = x_{ij} - x^0_{ij}$ and $x'_{ij}x'_{ji} = 0$, is feasible in the residual nonnegative graph $G(x^0) = (N, A(x^0))$. Furthermore, $cx = c'x' + cx^0$.*

The proof of the above theorem and related theory on the residual networks is well known. See, for example, Ahuja et al. (1993).

From Theorem 2.8, it can easily be verified that for each arc $(i, j) \in A$, the term $|x_{ij} - x^0_{ij}|$ that contributes to $f(x, x^0)$ is equal to $x'_{ij} + x'_{ji}$ in the residual graph. Therefore, $f(x, x^0) = \sum_{(i,j)} |x_{ij} - x^0_{ij}| = \sum_{(i,j)\in A(x^0)} x'_{ij} \leqslant k$. Combining this result with Theorem 2.8, we can formulate the incremental minimum cost flow problem P as a linear programming problem P^k as follows:

$$\min \sum_{(i,j)\in A(x^0)} c'_{ij}x_{ij} \tag{6a}$$

$$\text{subject to} \sum_{j:(i,j)\in A(x^0)} x_{ij} - \sum_{j:(j,i)\in A(x^0)} x_{ji} = 0 \quad \text{for all } i \in N, \tag{6b}$$

$$\sum_{(i,j)\in A(x^0)} x_{ij} \leqslant k, \tag{6c}$$

$$0 \leqslant x_{ij} \leqslant u'_{ij} \quad \text{for all } (i,j) \in A(x^0). \tag{6d}$$

Note that this is a minimum cost circulation problem with the additional constraint (6c) on the sum of the arc flows. Brucker (1985) shows how to solve this type of network flow problem using Lagrangian relaxation as a sequence of $O(\log nC)$ minimum cost flow problems, where $C = \max\{|c'_{ij}|\colon (i, j) \in A\}$.

2.2.2. Incremental Maximum Flow Problem. Let $G = (N, A)$ be a capacitated graph, s be a source node with a supply of d_s, and t be a sink node with a demand d_t. The amount $d = d_s = -d_t$ that is transferred from s to t by a flow x is called the flow value of the flow x. In a maximum flow problem, we maximize d through a flow $x \in X$, where X is the set of feasible flows defined by the mass-balance and capacity constraints (see Equation (5b)). We define the incremental maximum flow problem as follows:

Definition 2.9 (Incremental Maximum Flow). Let $G(N, A)$ be a capacitated graph and x^0 be an initial flow from a source node s to a sink node t with a flow value d. Find a flow x^* with a flow value d^* such that $d^* = \max\{d\colon f(x^0, x) \leqslant k, x \in X\}$, where $f(x^0, x) = \sum_{(i,j)\in A} |x_{ij} - x^0_{ij}|$.

We can define the incremental maximum flow problem as a special case of the incremental minimum cost circulation problem as follows. We assign a cost of 0 to every arc $(i, j) \in A$ and construct the residual graph $G(x^0) = (N, A(x^0))$ with respect to the initial flow x^0. We add an arc (t, s) with a cost of -1 and infinite capacity. We add the constraint, which limits the sum of the flow values on all of the arcs, excluding the additional arc (t, s), to be less than or equal to k. Finally, we minimize the cost of the circulation in the resulting graph. This problem is a minimum circulation problem with one additional constraint, and it is shown to have a polynomial time solution (Brucker 1985). In particular, it can be solved as a sequence of $O(\log n)$ minimum cost flow problems.

3. Other Incremental Network Optimization Problems

In this section, we present three other incremental network optimization problems. The first problem is the incremental shortest path problem. We study two versions of this problem depending on the increment function: (i) the arc inclusion version, and (ii) the arc exclusion version. We show that the first version can be solved efficiently and that the second version is NP-complete. The second problem is the incremental minimum cut problem, which we show to be NP-complete. The last problem is the incremental minimum assignment problem. We show that this problem reduces to the minimum exact matching problem, for which a randomized polynomial algorithm exists.

3.1. Incremental Shortest Path Problem

In an incremental shortest path problem, we are given an initial path P^0 from a node s to a node t in a directed network,

$G = (N, A)$, with arc costs c_{ij} for $(i, j) \in A$. Let P denote a path and the set of arcs in it, interchangeably, and let $c(P) = \sum_{(i,j)\in P} c_{ij}$ be the length (cost) of the path. We want to find a shortest possible path P^* from s to t such that $f(P^*, P^0) \leqslant k$, for a given increment function $f(\cdot, \cdot)$ and an integer k. In this section, we introduce two different versions of the incremental shortest path problem: (i) an arc inclusion version, and (ii) an arc exclusion version. We show that the arc inclusion version is polynomially solvable, whereas the arc exclusion version is NP-complete.

3.1.1. Incremental Shortest Path with Arc Inclusion. In the arc inclusion version of the incremental shortest path problem, the increment function gives the number of arcs that are in a path P and not in the initial path P^0; i.e., $f(P, P^0) = |P \backslash P^0|$. An incremental path P is such that $f(P, P^0) \leqslant k$. We can define the problem as follows:

Definition 3.1 (Incremental Shortest Path with Arc Inclusion). Let $G = (N, A)$ be a directed network and P^0 be an initial path from a node s to a node t. Find a path P^* such that $c(P^*) = \min\{c(P)\colon f(P, P^0) \leqslant k\}$, where $f(P, P^0) = |P \backslash P^0|$.

We show that ISP-I can be solved in polynomial time using dynamic programming recursions. Let

$d^r(j)$ denote the shortest path from node s to node j with at most r arcs in $A \backslash P^0$ and the last arc is in $A \backslash P^0$, and

$f^r(j)$ denote the shortest path from node s to node j with at most r arcs in $A \backslash P^0$.

Then, the following recursion solves the ISP-I problem:

$$d^r(j) = \min\{d^{r-1}(j), \min\{f^{r-1}(i) + c_{ij}\colon (i,j) \in A \backslash P^0\}\}, \quad (7)$$

$$f^r(j) = \min\{f^{r-1}(j), d^r(j), \min\{f^r(i) + c_{ij}\colon (i,j) \in P^0\}\}. \quad (8)$$

Note that for (7), $d^r(j)$ either

(i) has at most $r-1$ arcs in $A \backslash P^0$ and ends with an arc in $A \backslash P^0$, or

(ii) consists of a path with $r-1$ arcs in $A \backslash P^0$ followed by an arc in $A \backslash P^0$.

For (8), $f^r(j)$ either

(iii) has at most $r-1$ arcs in $A \backslash P^0$,

(iv) has at most r arcs $A \backslash P^0$ and the last arc is in $A \backslash P^0$, or

(v) has at most r arcs in $A \backslash P^0$, and the last arc is in P^0.

Condition (iii) holds although it is not mutually exclusive from conditions (iv) and (v).

Assume that nodes are ordered so that if node i precedes node j in P^0, then $i < j$. By definition, $d^0(j)$ can be set to ∞. It is easy to compute $f^0(j)$ because it is equal to the length of the segment of P^0 from node s to node j if $j \in P^0$, and ∞ if $j \neq P^0$. From (7) and (8), knowing $f^{r-1}(j)$ and $d^{r-1}(j)$ for all $j \in N$, we need only to check all of the incoming arcs $(i, j) \in A$ to node j to find $f^r(j)$ and $d^r(j)$. Thus, in the worst case, it takes $O(m)$ comparisons for each $r = 1, \ldots, k$. The following theorem summarizes the time complexity of our solution to ISP-I.

Theorem 3.2. *The incremental shortest path problem can be solved in $O(km)$ time.*

3.1.2. Incremental Shortest Path with Arc Exclusion. In the arc exclusion version of the incremental shortest path problem, the increment function gives the number of arcs that are in the initial path P^0 and are excluded in the new path P; i.e., $f(P, P^0) = |P^0 \backslash P|$; and an incremental path is such that $f(P, P^0) \leqslant k$. Now, we can define the problem as follows:

Definition 3.3 (Incremental Shortest Path with Arc Exclusion). Let $G = (N, A)$ be a directed network and P^0 be an initial path from node s to node t. Find a path P^* such that $c(P^*) = \min\{c(P)\colon f(P, P^0) \leqslant k\}$, where $f(P, P^0) = |P^0 \backslash P|$.

We show that the case $k = 1$ is polynomially solvable. Let $c(P)$ denote the cost of path P; let $P^0_{i,j}$ denote the segment of P^0 from node i to node j. Assume that we want to find a shortest path $P(i, j)$ from s to t, which excludes arc $(i, j) \in P^0$. Then, we can determine this path by finding the shortest path from i to j in the subgraph G', which excludes nodes in $P^0 \backslash \{i, j\}$ and excludes arc (i, j). Let this shortest path be denoted as $P^*_{i,j}$ and $P(i, j) = P^0_{s,i} - P^*_{i,j} - P^0_{j,t}$. Then, the solution is the path P^* such that $c(P^*) = \min\{c(P^0), \min\{c(P(i, j))\colon (i, j) \in P^0\}\}$.

Although the increment function of the arc exclusion version is slightly different from the arc inclusion version, in Theorem (3.4) we show that the case $k \geqslant 2$ is NP-complete by a reduction from the *2-directed disjoint paths (2DDP) problem*: Given a directed graph $G = (N, A)$ and distinct vertices s_1, s_2, t_1, t_2, find node-disjoint paths P_i from s_i to t_i for $i = 1, 2$. This problem is proved to be NP-complete by Fortune et al. (1980).

Theorem 3.4. *The problem of determining whether there is an incremental shortest path with a cost of 0 is NP-complete.*

Let $G = (N, E)$ be a directed graph in which we want to find node-disjoint paths P_1 from s_1 to t_1 and P_2 from s_2 to t_2. We transform G to a new graph G', on which we solve an ISP-E problem from s_1 to t_2 to solve 2-DDP, as follows:

1. Copy all of the nodes and arcs in G to G' and assign all of the arcs 0 cost.

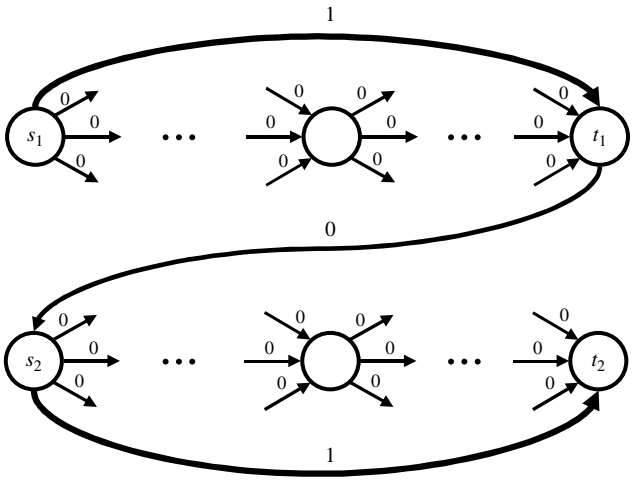
2. Add the arcs (s_1, t_1), (t_1, s_2), and (s_2, t_2) with costs of 1, 0, and 1, respectively.

3. Let the initial path be $P^0 = (s_1, t_1) - (t_1, s_2) - (s_2, t_2)$, and let $k = 2$.

The transformed graph G' is shown in Figure 2. The problem is whether we can find an incremental shortest path P^* in G' with 0 cost.

Note that there are three arcs in P^0. Only (t_1, s_2) has a cost of zero among these arcs. Let P^* be an incremental shortest path of 0 cost. Then, it is clear that (s_1, t_1) and (s_2, t_2) must be eliminated and (t_1, s_2) must be retained. Thus, $P^* = P^*_{(s_1,t_1)} - (t_1, s_2) - P^*_{(s_2,t_2)}$, where all the arcs in $P^*_{(s_1,t_1)}$ and $P^*_{(s_2,t_2)}$ are also in G. Because P^* is a path, by definition $P^*_{(s_1,t_1)}$ and $P^*_{(s_2,t_2)}$ are node disjoint, and they constitute a solution to 2-DDP: $P_1 = P^*_{(s_1,t_1)}$ and $P_2 = P^*_{(s_2,t_2)}$.

Figure 2. The bold arcs are added for the transformation from G to G'.



Conversely, assume that we have node-disjoint paths P_1 from s_1 to t_1 and P_2 from s_2 to t_2 in G, which is a solution to the 2-DDP problem. Then, $P_1 - (t_1, s_1) - P_2$ is an incremental path from s_1 to t_2 of cost 0. We note that we can extend the proof for the general case $k \geqslant 2$ by replacing the arc (s_2, t_2) by a path with $k-1$ arcs, each with cost 0.

3.2. Incremental Minimum Cut

In a directed network $G = (N, A)$, a partition of the set of nodes N into two sets S and $\bar{S}$ is called a *cut* and is denoted as $[S, \bar{S}]$. Let s be the source node and t be the sink node; we call a cut an $s-t$ *cut* if $s \in S$ and $t \in \bar{S}$. Let $\{S, \bar{S}\}$ denote the set of arcs whose tail nodes are in S and head nodes are in $\bar{S}$; i.e., $\{S, \bar{S}\} = \{(i, j)\colon i \in S, j \in \bar{S}\}$. The capacity of a cut $u[S, \bar{S}]$ is defined as the sum of the capacities u_{ij} of all the arcs in $\{S, \bar{S}\}$; i.e., $u[S, \bar{S}] = \sum_{(i,j)\in\{S,\bar{S}\}} u_{ij}$. Let $[S^0, \bar{S}^0]$ be the initial cut and $[S, \bar{S}]$ be any other cut. Then, the increment function, used as a measure of the difference between two cuts, is defined as $f(S, S^0) = f(\bar{S}, \bar{S}^0) = |S \oplus S^0|$, which is the cardinality of the exclusive union of the sets S and S^0. We give the definition of the incremental minimum cut problem below.

Definition 3.5 (Incremental Minimum Cut). Given a directed graph $G = (N, A)$ with a source node s, a sink node t, and an initial $s-t$ cut $[S^0, \bar{S}^0]$, find an $s-t$ cut $[S^*, \bar{S}^*]$ so that $u[S^*, \bar{S}^*] = \min\{u[S, \bar{S}]\colon f(S, S^0) \leqslant k\}$, where $f(S, S^0) = |S \oplus S^0|$.

We show that this problem is NP-complete by a reduction from the *graph partition problem*: Given an undirected graph $G' = (V, E)$ with an even number of nodes, is there a partition of the vertex set V into $\mathcal{V}$ and $\overline{\mathcal{V}}$ such that $|\mathcal{V}| = |\overline{\mathcal{V}}| = |V|/2$, and $|\{\mathcal{V}, \overline{\mathcal{V}}\}| \leqslant K$, where $\{\mathcal{V}, \overline{\mathcal{V}}\} = \{(i, j)\colon i \in \mathcal{V},\ j \in \overline{\mathcal{V}}\}$? This problem was proved to be NP-complete by Hyafil and Rivest (1973). We will reduce an instance of the graph partition problem to an incremental minimum cut problem, where $k = n/2$.

Theorem 3.6. *The problem of determining an incremental cut with capacity at most K' is NP-complete.*

Let $G' = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$ on which we want to solve a graph partition problem. We transform G' into $G = (N, A)$ as follows:

1. Replace all undirected arcs $(i, j) \in E$ with two directed arcs (i, j) and (j, i) each with a capacity of 1.
2. Add a source node s and arcs $\{(s, i)\colon i \in V\}$ with capacities of $m + 1$.
3. Add a sink node t and arcs $\{(i, t)\colon i \in V\}$ with capacities of 0.
4. Let the initial cut $[S, \bar{S}]$ be such that $S = \{s\}$ and $\bar{S} = V \cup \{t\}$ with $u[S, \bar{S}] = n(m + 1)$.
5. Let $k = n/2$ and $K' = (n/2)(m + 1) + K$.

Suppose first that $[R, \bar{R}]$ is an incremental cut in G' with capacity at most K'. It is clear that $|R| \leqslant n/2 + 1$. If $|R| < n/2$, then

$$u[R, \bar{R}] \geqslant (m + 1)(1 + n/2) > K'. \tag{9}$$

So, if $u[R, \bar{R}] \leqslant K'$, it follows that $|R| = n + 1$. In this case,

$$\begin{aligned} u[R, \bar{R}] &= (n/2)(m + 1) + u[R\backslash\{s\}, \bar{R}\backslash\{t\}] \\ &\leqslant (n/2)(m + 1) + K, \end{aligned}$$

and so there is a feasible solution for the partition problem.

Conversely, suppose that $[V_1, V_2]$ is a solution to the graph partition problem. Then, $[V_1 \cup \{s\}, V_2 \cup \{t\}]$ is an incremental cut with capacity at most K', completing the proof.

3.3. Incremental Minimum Assignment Problem

In an undirected graph $G = (N, A)$, a subset of arcs $A' \subseteq A$ such that every node has an incident arc is called a *perfect matching*. If G is a bipartite graph, then a perfect matching is called an *assignment*. Let $c(i, j)$ denote the cost of arc (i, j). For an assignment A', its cost $c(A')$ is the sum of all of the arc costs in A'; i.e., $c(A') = \sum_{(i,j)\in A'} c(i, j)$. Let A^0 be an initial assignment in G, and let the increment function for an assignment be defined as $f(A, A^0) = |A\backslash A^0|$, which is the number of arcs in the new assignment A that are not in the original assignment A^0. We can define the incremental minimum assignment problem as follows:

Definition 3.7 (Incremental Minimum Assignment). Let $G = (N, A)$ be a directed bipartite graph and B^0 be an initial assignment in G. Find an assignment B^* such that $c(B^*) = \min\{c(B)\colon f(B, B^0) \leqslant k\}$, where $f(B, B^0) = |B\backslash B^0|$.

We show that the incremental minimum assignment problem can be reduced to the *minimum exact matching problem*: Given a bipartite graph $G = (N, A)$, a subset of red arcs $R \subseteq A$, and an integer t, find a perfect matching B^t with exactly t red arcs such that $c(B^t)$ is the minimum possible.

Lemma 3.8. *The incremental minimum assignment problem is a special case of the minimum exact matching problem.*

In the incremental minimum assignment problem, the arcs that are not in the initial assignment B^0 can be considered as the set of red arcs; i.e., $R = A \backslash B^0$. If we add n additional arcs that are copies of arcs in A^0, then we may assume without generality that the optimum solution B^* has $f(B^*, B^0) = k$ because we can replace arcs of A^0 with their copies and increment $f(B^*, A^0)$. We can find a solution to the incremental minimum assignment problem by solving the minimum exact matching problem for each t.

The minimum exact matching problem is solved by Mulmuley et al. (1987) in random polynomial time if the data are encoded in unary or if there is an upper bound on the cost coefficients that is polynomial in n.

We note that if we measured the increment function by arc exclusion, there would be no difference: For any two assignments A and A^0, $|A \backslash A^0| = |A^0 \backslash A|$.

4. Conclusion

In this paper, we study six incremental network optimization problems. We show that, using Lagrangian relaxation and approximate binary search, we can efficiently solve the incremental minimum spanning tree problem. We also show that the incremental minimum cost flow and incremental maximum flow problems are polynomial time solvable using Lagrangian relaxation. We study two versions of the incremental shortest path problem. The arc inclusion version, which limits the number of new arcs, can be solved very efficiently, whereas the arc exclusion version, which limits the number of arcs excluded from the original solution, is an NP-complete problem. We show that the incremental minimum cut problem is also NP-complete and that the incremental minimum assignment problem is a special case of the minimum exact matching problem, which can be solved by a randomized polynomial time algorithm.

Acknowledgments

The third author gratefully acknowledges partial support from a National Science Foundation grant and from Office of Naval Research grant N000140810029.

References

Aarts, E., J. K. Lenstra, eds. 1997. *Local Search in Combinatorial Optimization*, 1st ed. John Wiley & Sons, New York.

Adomavicius, G., A. Gupta. 2005. Toward comprehensive real-time bidder support in iterative combinatorial auctions. *Inform. Systems Res.* **16**(2) 169–185.

Ahuja, R. K., K. Jha, J. Liu. 2007. Solving real-life railroad blocking problems. *Interfaces* **37**(5) 404–419.

Ahuja, R. K., T. M. Magnanti, J. B. Orlin. 1993. *Network Flows*: *Theory, Algorithms, and Applicatons*. Prentice Hall, Upper Saddle River, NJ.

Ahuja, R. K., Ö. Ergun, J. B. Orlin, A. P. Punnen. 2002. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123**(1–3) 75–102.

Ahuja, R. K., P. Dewan, M. Jaradat, K. C. Jha, A. Kumar. 2006. An optimization-based decision support system for train scheduling. Technical report, Innovative Scheduling, Gainesville, FL.

Brucker, P. 1985. Parametric programming and circulation problems with one additional linear constraint. H. Noltemeier, ed. *Proc. WG'85*, *Workshop on Graph-Theoretic Concepts in Computer Science*, Wurzburg, Germany, 12–21.

Chazelle, B. 2000. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM* **47**(6) 1028–1047.

Dimitromanolakis, A. 2002. An analysis of the Golomb ruler and the Sidon set problems, and determination of large near-optimal Golomb rulers. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, Crete, Greece.

Dos Santos, B. L., M. L. Bariff. 1988. A study of user interface aids for model-oriented decision support systems. *Management Sci.* **34**(4) 461–468.

Fortune, S., J. Hopcroft, J. Wyllie. 1980. The directed homeomorphism problem. *Theoretical Comput. Sci.* **10** 111–121.

Higgins, P. G. 1995. Interactive job-shop scheduling: How to combine operations research heuristics with human abilities. *6th Internat. Conf. Manufacturing Engrg.*, Institution of Engineers, Melbourne, Victoria, Australia, 293–302.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*: *An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Hyafil, L., R. L. Rivest. 1973. Graph partitioning and constructing optimal decision trees are polynomial complete problems. Report 33, IRIA Laboria, Rocquencourt, France.

Li, G., S. Rajagopalan. 1998. Process improvement, quality, and learning effects. *Management Sci.* **44**(11) 1517–1532.

Little, J. D. C. 1970. Models and managers: The concept of a decision calculus. *Management Sci.* **16** B466–B485.

Mulmuley, K., U. V. Vazirani, V. V. Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* **7**(1) 105–113.

Regan, P. J. 2006. Professional decision modeling: Practitioner as professor. *Interfaces* **36**(2) 142–149.

Vaidyanathan, B., R. K. Ahuja, K. C. Jha. 2008. Real-life locomotive planning: New formulations and computational results. *Transportation Res. B* **42**(2) 147–168.

Vaidyanathan, B., K. C. Jha, R. K. Ahuja. 2007. Multicommodity network flow approach to the railroad crew scheduling problem. *IBM J. Res. Development* **51**(3/4) 325–344.

Wagner, H. M. 1977. *Principles of Operations Research*. Prentice-Hall, Englewood Cliffs, NJ.

Zangwill, W. I., P. B. Kantor. 1998. Toward a theory of continuous improvement and the learning curve. *Management Sci.* **44**(7) 910–920.