Ann Oper Res (2007) 153: 257–296
DOI 10.1007/s10479-007-0172-6

Complexity and algorithms for nonlinear optimization problems

Dorit S. Hochbaum

Published online: 3 May 2007

Abstract Nonlinear optimization algorithms are rarely discussed from a complexity point of view. Even the concept of solving nonlinear problems on digital computers is not well defined. The focus here is on a complexity approach for designing and analyzing algorithms for nonlinear optimization problems providing optimal solutions with prespecified accuracy in the solution space. We delineate the complexity status of convex problems over network constraints, dual of flow constraints, dual of multi-commodity, constraints defined by a submodular rank function (a generalized allocation problem), tree networks, diagonal dominant matrices, and nonlinear knapsack problem's constraint. All these problems, except for the latter in integers, have polynomial time algorithms which may be viewed within a unifying framework of a *proximity-scaling* technique or a *threshold* technique. The complexity of many of these algorithms is furthermore best possible in that it matches lower bounds on the complexity of the respective problems.

In general nonseparable optimization problems are shown to be considerably more difficult than separable problems. We compare the complexity of continuous versus discrete nonlinear problems and list some major open problems in the area of nonlinear optimization.

Keywords Nonlinear optimization · Convex network flow · Strongly polynomial algorithms · Lower bounds on complexity

1 Introduction

Nonlinear optimization problems are considered to be harder than linear problems. This is the chief reason why approximate linear models are frequently used even if the circumstances justify a nonlinear objective. A typical approach is to replace an objective function

An earlier version of this paper appeared in *4OR*, 3:3, 171–216, 2005.

D.S. Hochbaum (✉)
Department of Industrial Engineering and Operations Research and Walter A. Haas School of Business, University of California, Berkeley, USA
e-mail: hochbaum@ieor.berkeley.edu

that is nonlinear by a piecewise linear function. This approach may adversely affect the algorithm's complexity as often the number of pieces is very large and integer variables may have to be introduced. For problems with nonlinear objective function the leading methodology consists of iterative and numerical algorithms in which complexity analysis is substituted with a convergence rate proof.

In order to apply complexity analysis to nonlinear optimization problems, it is necessary to determine what it means to solve such a problem. Unlike linear problems, for nonlinear problems the length of the output can be infinite, such as in cases when a solution is irrational. There are two major complexity models for nonlinear optimization. One that seeks to approximate the objective function. The second, which we present here, approximates the optimal solution in the solution space. The latter has a number of advantages described next in Sect. 1.1.

Our goals here are to set a framework for complexity analysis of nonlinear optimization with linear constraints, delineate as closely as possible the complexity borderlines between classes of nonlinear optimization problems, and generate a framework of effective techniques. In particular it is shown how properties of convexity, separability and quadraticness of nonlinear optimization contribute to a substantial reduction in the problems' complexity. We review a spectrum of techniques that are most effective for each such class and demonstrate that in some cases the techniques lead to best possible algorithms in terms of their efficiency.

This paper is an updated version of Hochbaum (2005). Some of the subjects covered appeared earlier in Hochbaum (1993).

1.1 The complexity model, or what constitutes a solution to a nonlinear optimization problem

There is a fundamental difficulty in solving nonlinear optimization on digital computers. Unlike linear programming, for which all basic solutions to a set of linear inequalities require only finite accuracy of polynomial length in the length of the input (see, e.g., Papadimitiou and Steiglitz 1982, Lemma 2.1), one cannot bound a priori the number of digits required for the length of nonlinear programming optimal solutions. This feature is important since computers can only store numbers of finite accuracy. Even the simplest nonlinear optimization problems can have irrational solutions and thus writing the output alone requires infinite complexity. This is the case, for instance, in the minimization of the convex function $\max\{x^2 - 2, 0\}$. So the interpretation of what it means to solve a problem in reals is not obvious.

Traditional techniques for coping with nonlinear problems are reviewed extensively in Minoux (1986). These techniques have several shortcomings. In some applications, the nonlinear function is not available analytically. It is accessible via a data acquisition process or as a solution to a system of differential equations (as in some queuing systems). A typical traditional method approximates first the data input function as an analytic function, while assuming it is a polynomial of a conveniently low degree. Consequently, there are errors attributed to the inaccuracy of the assumed input even before an algorithm is applied for solving the problem.

Further difficulties arise because of conditions required for the algorithms to work. The algorithms typically make use of information about the function's derivatives from numerical approximations, which incorporates a further element of error in the eventual outcome. Moreover, certain assumptions about the properties of the nonlinear functions, such as differentiability and continuity of the gradients, are often made without any evidence that the "true" functions indeed possess them.

In addition to the computational difficulties, the output for nonlinear continuous optimization problems can consist of irrational or even transcendental (non algebraic) numbers. Since there is no finite representation of such numbers, they are usually truncated to fit within the prescribed accuracy of the hardware and software.

Complexity analysis requires finite length input and output which is of polynomial length as a function of the length of the input. Therefore the usual presentation of nonlinear problems renders complexity analysis inapplicable. For this reason complexity theory has not addressed nonlinear problems, with the exception of some quadratic problems. To resolve this issue, the nonlinear functions can be given in form of a table or *oracle*. An oracle accepts arguments of limited number of digits, and outputs the value of the function truncated to the prescribed number of digits.

Since nonlinear functions cannot be treated with the same absolute accuracy as linear functions, the notion of approximate solutions is particularly important. One definition of a *solution* to a nonlinear optimization problem is one that approximates the objective function. Among the major proponents of this approach are Nemirovsky and Yudin (1983) who chose to approximate the objective value while requiring some prior knowledge about the properties of the objective function. We observe that any information about the behavior of the objective at the optimum can always be translated to a level of accuracy of the solution vector itself (and vice versa). A detailed discussion of this point is provided in Hochbaum and Shanthikumar (1990). We thus believe that the interest of solving the optimization problem is in terms of the accuracy of the solution rather than the accuracy of the optimal objective value.

We thus use a second definition of a *solution* is one that approximates the solution in the solution space, within a prescribed accuracy of ϵ. According to the concept of ϵ-accuracy of Hochbaum and Shanthikumar (1990) a solution is said to be ϵ-accurate if it is at most at a distance of ϵ (in the L_∞ norm) from an optimal solution. That is, a solution, $\mathbf{x}^{(\epsilon)}$ is *ϵ-accurate* if there exists an optimal solution $\mathbf{x}^*$ such that $||\mathbf{x}^{(\epsilon)} - \mathbf{x}^*||_\infty \leq \epsilon$. ϵ is then said to be the accuracy required in the solution space. In other words, the solution is identical to the optimum in $O(\log \frac{1}{\epsilon})$ decimal digits.

Both definitions of a solution overlap for nonlinear problems on integers in which case an accuracy of $\epsilon < 1$ is sufficient.

The computation model here assumes the *unit cost model*, i.e. any arithmetic operation or comparison is counted as a single operation, even if the operands are real numbers. We use here however only operands with up to $O(\log \frac{1}{\epsilon})$ significant digits.

A similar complexity model using ϵ-accuracy was used by Shub and Smale (1996); Renegar (1987) and others. These works, however, address algebraic problems, as opposed to optimization problems addressed here.

1.2 Classes of nonlinear problems addressed and some difficult cases

A fundamental concept linking between the solutions to continuous and integer problems is that of *proximity*. Classes of convex optimization problems for which there is a "good" proximity with relative closeness of the continuous and integer solutions are solvable in polynomial time that depends on that distance. These classes include problems with a constraint matrix that has small subdeterminants. A prominent example of such optimization problems is the convex separable network flow problem. Other classes with a good proximity addressed here are the problems with *polymatroidal* constraints (the allocation problem discussed in Sect. 5) and the NP-hard nonlinear knapsack problem. For the nonlinear knapsack problem it is shown that approaches relying on proximity are effective in generating

a fully polynomial approximation scheme for the nonlinear knapsack problem as well as other results equivalent to those that hold for the linear knapsack problem. That is, the nonlinearity of the objective function does not make the problem any harder.

Among nonlinear problems with a constraint matrix that has very large subdeterminants, we discuss here the inverse shortest paths problem that has a constraint matrix the same as the dual of the multicommodity flow problem. For that problem we employ a proximity theorem called *projected proximity* that allows to get polynomial time algorithms for this class of nonlinear problems.

Convex problems are discussed here with the classification: separable or nonseparable; quadratic or non-quadratic; integer or continuous. The *nonconvex continuous* version of the separable minimum cost network flow problem is NP-hard Sahni (1974) even for the quadratic case. The corresponding concave minimization problem is in general NP-hard (see, e.g., Guisewite and Pardalos 1990). An excellent unifying presentation of some polynomial instances of the concave separable cost minimization flow problem is given in Erickson et al. (1987). There the problem is proved to be polynomial when the arcs are incapacitated and number of demand nodes is fixed. The problem is also proved to have polynomial time algorithms for certain classes of planar graphs.

The *nonseparable* nonlinear optimization problem is in general hard. Even with the assumption of convexity, a quadratic nonseparable problem is NP-hard (see Sect. 11.) In spite of these negative results, there are a number of subclasses of practical interest that are solvable in polynomial time, which we point out.

As a general rule, and as we show later, a convex separable nonquadratic integer flow problem is easier to solve than the respective continuous one, using a proximity result. The exception to this rule is the *quadratic* convex problems for which it is typically easier to obtain continuous solutions rather than integer ones. In that case an integer optimum is derived from the continuous solution instead of the other way around.

1.3 Polynomial algorithms and issues of strong polynomiality

All polynomial time algorithms presented here, and indeed all algorithms known, that solve nonlinear nonquadratic problems and run in polynomial time, do not have a *strongly polynomial* complexity. That is, the running time depends on the magnitude of some "number" in the data which is typically the range of the interval in which the variables are bounded.

For example, the most efficient algorithms known to date for the convex integer separable flow problem are based on the concept of scaling. These include an algorithm by Minoux (1986), an algorithm by Ahuja et al. (1993) and an algorithm of "proximity-scaling" type by Hochbaum and Shanthikumar (1990) (presented in Sect. 3). The complexity of the proximity-scaling algorithm for the integer problem is $O(\log \frac{B}{m}(m+n)(m+n\log n))$ where B is the largest capacity or supply in the network and n, m, the number of nodes and arcs respectively. For the continuous problem the ϵ-accurate solution is derived in $O(\log \frac{B}{\epsilon}(m+n)(m+n\log n))$ steps.

These three algorithms are polynomial but not strongly polynomial, as they depend, via the quantity B, on the magnitude of the numbers appearing in the problem instance, as well as on n and m. A naturally arising question is whether it is possible to devise an algorithm the complexity of which is independent of B and dependent only on n and m, that is, a strongly polynomial algorithm. This question was answered in the negative with an impossibility result for strongly polynomial algorithms in Hochbaum (1994) (Sect. 2).

Strong polynomiality has emerged as an important issue since the first polynomial time algorithm, the ellipsoid method, was devised for solving linear programming problems. The

ellipsoid method, as well as all other polynomial algorithms known for linear programming, runs polynomial but not strongly polynomial time. That is, the running time depends on the data coefficients rather than only on the number of variables and constraints. Consequently, solving linear programming problems (and other problems that do not possess strongly polynomial algorithms) with different degrees of accuracy in the cost coefficients results in different running times. So the actual number of arithmetic operations grows as the accuracy of the data, and hence the length of the numbers in the input, increases. Such behavior of an algorithm is undesirable as it requires careful monitoring of the size of the numbers appearing in the data describing the problem instance, and thus limits the efficient applicability of the algorithm.

Although it is not known whether linear programming can be solved in strongly polynomial time, Tardos (1986) established that "combinatorial" linear programming problems, those with a constraint matrix having small coefficients, are solvable in strongly polynomial time. Thus, minimum (linear) cost network flow problems are solvable in strongly polynomial time (Tardos 1985) since their constraint matrix' coefficients are either 0, 1 or -1. In contrast, nonlinear and non-quadratic 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 Hochbaum (1994). So while convex separable minimization is solved in polynomial time on totally unimodular matrices (Hochbaum and Shanthikumar 1990), linear optimization on such constraint matrices runs in strongly polynomial time.

This negative result 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 have been shown 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. A number of special cases of the minimum quadratic cost network flow problem that are solvable in strong polynomial time are reviewed in Sect. 10.

1.4 Overview

We begin with the impossibility result and lower bound on the complexity of nonlinear problems in Sect. 2. The lower bound provided applies in both the comparison model and in the algebraic tree model. Section 3 describes the proximity-scaling algorithm for convex separable optimization problem with constraint matrices that have bounded subdeterminants. We focus on the interpretation of the technique as a form of piecewise linear approximation of nonlinear functions that uses specific scaling so as to guarantee polynomial complexity. A specific implementation of the approach to convex network flow is given in Sect. 4. In Sect. 5 we describe the use of a proximity-scaling approach to the general allocation problem and its special cases. Here the proximity theorem used is stronger than the one for general problems on linear constraints. The algorithms generated are shown to be best possible for most classes of the allocation problem. The use of proximity to reduce the nonlinear knapsack problem to an allocation problem is described in Sect. 6. This leads to a polynomial time algorithm for the nonlinear continuous knapsack, which in turn makes a fully polynomial time approximation scheme available for the nonlinear knapsack problem. The use of proximity-scaling for the convex dual of minimum cost network flow is sketched next in Sect. 7. The proximity-scaling approach is concluded in Sect. 8 where a "projected proximity" theorem is shown to be applicable to the convex dual of the multi-commodity flow problem, with application to inverse shortest paths problem.

The next leading technique we describe is based on "threshold theorems" in Sect. 9. It is shown there how this technique is used for the convex cost closure and the convex s-excess problem with one application, among many others, to the image segmentation problem.

Classes of quadratic problems known to be solved in strongly polynomial time are delineated in Sect. 10. Various classes of nonseparable cases that are solved efficiently, and a collection of relevant techniques, are given in Sect. 11. Section 12 contains some concluding remarks and lists open problems.

Notation used in this paper includes bold letters for denoting vectors, and $\mathbf{e}$ to denote the vector $(1, 1, \ldots, 1)$. The vector $\mathbf{e}^j$ has 0s in all positions except position j that is equal to 1. When discussing a network the standard notation of $G = (V, A)$ is used with the number of nodes $|V|$ denoted by n and the number of arcs $|A|$ denoted by m. In the discussion on nonlinear programming the number of variables is denoted by n and the number of constraints (excluding nonnegativity) by m.

2 The impossibility of strongly polynomial algorithms for convex separable network flow problems

An impossibility result on the existence of a strongly polynomial algorithm for nonlinear problems was proved in Hochbaum (1994) and reviewed here. This result applies to convex separable minimization, and even to constraint matrices that are very simple, such as network flow constraints or even a single constraint bounding the sum of the values of the variables. This lower bound holds in the comparison model for all nonlinear problems. In a more general model—the *algebraic-tree* model that permits all the arithmetic operations—strongly polynomial algorithms are provably impossible for nonlinear and nonquadratic algorithms. That leaves open the possibility of strongly polynomial algorithms for quadratic convex separable minimization problems.

The problem for which the lower bound is given is the *simple resource allocation problem*. The simple resource allocation problem (denoted by the acronym SRA) is identical to a single source transportation problem in maximization form:

$$\text{(SRA)} \quad \max \quad \left\{ \sum_{i=1}^{n} f_i(x_i) \;\Big|\; \sum_{i=1}^{n} x_i = B,\ \mathbf{x} \geq 0 \right\}.$$

The generic presentation of this problem is as a concave maximization problem (with an obvious translation to the minimization/convex case). We first present a comparison model lower bound followed by an algebraic tree model lower bound.

2.1 A comparison model lower bound

A comparison computation model allows only the operations of comparisons and branchings. The lower bound proof establishes that no algorithm exists that solves SRA in less than $\log_2 B$ comparisons. To show that, we rely on a result of information theory according to which there is no algorithm that finds a value in a monotone nonincreasing n-array that is the first to be smaller than some specified constant in less than $\log_2 n$ time.

Consider first a lower bound result for SRA in two variables, SRA(2):

$$\begin{aligned} \text{(SRA(2))} \quad \max \quad & f_1(x_1) + cx_2 \\ & x_1 + x_2 = B, \\ & x_1, x_2 \geq 0, \quad \text{integer}. \end{aligned}$$

Let the function $f_1(x_1)$ be given as an array of B increments at the B integer points. Namely, if $f_1(i) = a_i$ the array of increments is $\{a_0, a_1 - a_0, a_2 - a_1, \ldots, a_B - a_{B-1}\}$. Since the function is concave, the entries in the array are monotone nonincreasing, $a_{i+1} - a_i \leq a_i - a_{i-1}$. The optimal solution to this problem is $x_1 = j$ and $x_2 = B - j$, where j is the largest index such that $a_j - a_{j-1} \geq c$.

Since the array of the differences between consecutive entries of the array is monotone nonincreasing, determining in the array of differences the index j can be done using binary search in $\log_2 B$ comparisons. The information theoretic lower bound is also $\log_2 B$ comparisons. This is because the comparison tree has B leaves so the path of comparisons leading from the root to a leaf could be as long as $\log_2 B$ (see Knuth 1973).

Suppose the problem could be solved independently of B, then given a monotone nonincreasing array and a value c, it has a corresponding concave function, f_1 such that the solution to the SRA(2) is independent of B. Consequently, the required entry in the vector could be found independently of B, which is a contradiction to the comparison tree lower bound.

A similar proof works for the problem with a single variable if the constraint is an inequality constraint.

The same arguments can be extended to prove that in the comparison model the allocation problem on $n + 1$ variables has complexity $\Omega(n \log_2 \frac{B}{n})$. Let the problem be defined for $c > 0$:

$$\begin{aligned} (\mathrm{SRA}(n+1)) \quad \max \quad & \sum_{j=1}^{n} f_j(x_j) + c \cdot x_{n+1} \\ & \sum_{j=1}^{n+1} x_j = B, \\ & x_j \geq 0, \quad \text{integer}, \ j = 1, \ldots, n+1. \end{aligned}$$

Let the functions f_j be concave and monotone increasing in the interval $[0, \lfloor \frac{B}{n} \rfloor]$, and zero in $[\lfloor \frac{B}{n} \rfloor, B]$. Solving SRA$(n+1)$ is then equivalent to determining in n arrays of length $\lfloor \frac{B}{n} \rfloor$ each, the last entry of value $\geq c$. Since the arrays are independent, the information theory lower bound is $\Omega(n \log \lfloor \frac{B}{n} \rfloor)$. Similarly, for the case of an inequality constraint the same lower bound applies for the problem on n variables, since x_{n+1} can simply be viewed as the slack and $c = 0$.

This comparison model lower bound holds also for the quadratic case. It is therefore impossible to solve the quadratic problems in strongly polynomial time using only comparisons.

Indeed the floor operation is essential for the quadratic integer problem and without it there is no hope of solving the integer version in strongly polynomial time This follows from an observation by Tamir (1993), that demonstrated this via the following quadratic allocation problem.

$$\begin{aligned} \min \quad & \left[\frac{1}{2} x_1^2 + \frac{1}{2}(a-1) x_2^2 \right], \\ \text{s.t.} \quad & x_1 + x_2 = b, \\ & x_1, x_2 \geq 0, \quad \text{integer}. \end{aligned}$$

The optimal value of x_2 is $\lfloor \frac{B}{a} \rfloor$. Therefore the floor operation can be executed via a routine that solves a quadratic allocation problem. We demonstrate in the next section an impossibility result of strongly polynomial algorithms for non-quadratic problems which implies

that the floor operation, if helpful in devising strongly polynomial algorithms, is limited in its power to quadratic problems only.

2.2 The algebraic-tree model lower bound

One might criticize the choice of the comparison model as being too restrictive. Indeed, the use of arithmetic operations may help reduce the problems complexity. This is the case for the quadratic SRA, which is solvable in linear time, $O(n)$ (Brucker 1984). The lower bound here demonstrates that such success is not possible for other nonlinear functions (non-quadratic). The algebraic-tree computation model permits the arithmetic operations $+, -, \times, \div$ as well as comparisons and branchings based on any of these operations. It is demonstrated that the nature of the lower bound is unchanged even if the floor operation is permitted as well.

We rely on Renegar's lower bound proof (Renegar 1987) in this arithmetic model of computation for finding ϵ-accurate roots of polynomials of fixed degree ≥ 2. In particular, the complexity of identifying an ϵ-accurate single real root in an interval $[O, R]$ is $\Omega(\log\log \frac{R}{\epsilon})$ even if the polynomial is monotone in that interval. Let $p_1(x), \ldots, p_n(x)$ be n polynomials each with a single root to the equation $p_i(x) = c$ in the interval $[0, \frac{B}{n}]$, and each $p_i(x)$ a monotone decreasing function in this interval. Since the choice of these polynomials is arbitrary, the lower bound on finding the n roots of these n polynomials is $\Omega(n \log\log \frac{B}{n\epsilon})$. Let $f_j(x_j) = \int_0^{x_j} p_j(x)\,dx$. The $f_j s$ are then polynomials of degree ≥ 3. The problem,

$$
\begin{aligned}
(P_\epsilon) \quad \max \quad & \sum_j f_j(x_j \cdot \epsilon) + c \cdot x_{n+1} \cdot \epsilon \\
& \sum_{j=1}^{n+1} x_j = \frac{B}{\epsilon}, \\
& x_j \geq 0 \quad \text{integer}, \ j = 1, \ldots, n+1
\end{aligned}
$$

has an optimal solution $\mathbf{x}$ such that $\mathbf{y} = \epsilon \cdot \mathbf{x}$ is the $(n\epsilon)$-accurate vector of roots solving the system

$$
\begin{cases}
p_1(y_1) = c, \\
p_2(y_2) = c, \\
\vdots \\
p_n(y_n) = c.
\end{cases}
$$

This follows directly from the Kuhn–Tucker conditions of optimality and the proximity theorem to be discussed in Sect. 3.2, that an optimal integer solution $\mathbf{x}^*$ to the scaled problem with a scaling constraint s and the optimal solution to the continuous problem $\mathbf{y}^*$ satisfy $\|\mathbf{x}^* - \mathbf{y}^*\|_\infty \leq ns$ (Theorem 1). Hence, a lower bound for the complexity of solving (P_ϵ) is $\Omega(n \log\log \frac{B}{n\epsilon})$. For $\epsilon = 1$, we get the desired lower bound for the integer problem.

Mansour et al. (1991) proved a lower bound on finding ϵ-accurate square roots that allows also the floor, $\lfloor\ \rfloor$, operation. In our notation this lower bound is $\Omega(\sqrt{\log\log \frac{B}{\epsilon}})$. Hence, even with this additional operation the problem cannot be solved in strongly polynomial time. Again, the quadratic objective is an exception and indeed algorithms for solving the quadratic objective SRA problem rely on solving the continuous solution first, then rounding down, using the floor operation, and proceeding to compute the resulting integer vector to feasibility and optimality using fewer than n greedy steps. See for instance Ibaraki and Katoh

(1988) for such an algorithm. Since the lower bound result applies also in the presence of the floor operation, it follows that the "ease" of solving the quadratic case is indeed due to the quadratic objective and not to this, perhaps powerful, operation.

3 A polynomial proximity-scaling algorithm

The proximity-scaling algorithm makes use of the fact that the convex piecewise linear minimization problem on a "small" number of pieces is solvable in polynomial time (Dantzig 1963 and Lemma 1). The proximity-scaling algorithm is based on approximating the convex functions by piecewise linear functions on a uniform grid with a small number of breakpoints. The proximity theorems state that the solution to such scaled problems is close to the optimal solution in the solution space, thus allowing to update the length of the interval in which each variable lies by a constant factor. A logarithmic number of calls to solving the scaled piecewise linear function then leads to an optimal solution in integers, or within ϵ-accuracy.

While the proximity-scaling procedure of Hochbaum and Shanthikumar (1990) provides a polynomial time algorithm for any convex separable optimization problem on totally unimodular constraints (or for problems with constraint matrices that have bounded subdeterminants), it is shown next that a specialized implementation taking into account the network structure and the equality balance constraints, is more efficient than the general purpose algorithm.

3.1 The scaled piecewise linear approximation

The idea of (piecewise) linearizing a nonlinear function in order to obtain solutions has been well known. In a 1959 book Dennis (1959) writes regarding quadratic cost networks:

> The electrical model for network flow problems can be extended to include flow branches for which the total cost contains terms depending on the square of the individual branch flows .... It appears that the algorithms presented in this chapter could be generalized .... These methods however are combinatorial in character and could require prohibitive calculation time, even on relatively simple networks. Certainly the simplicity and elegance of the diode-source algorithms would be absent. It would seem that the most practical means of attacking flow problems with quadratic costs would be to approximate the cost curve with piece-wise linear curve and substitute an appropriate number of linear cost branches connected in parallel.

Whereas it is clear that solving the problem on a piecewise linear approximation yields a feasible solution, the quality of such solution and its closeness to an optimal solution were not evaluated until the work of Hochbaum and Shanthikumar (1990).

We consider a convex separable minimization problem

$$\min\{F(\mathbf{x} \mid T\mathbf{x} = \mathbf{b},\ \ell \le \mathbf{x} \le \mathbf{u}\}.$$

The scaling process is illustrated for the convex network flow problem. For a network $G = (V, A)$ the variables are $\mathbf{x} = \{x_{ij}\}_{(i,j)\in A}$ and $F(\mathbf{x}) = \sum_{(i,j)\in A} f_{ij}(x_{ij})$.

For a scaling constant s, the piecewise linearized objective function is $F^s(\mathbf{x}) = \sum_{(i,j)\in A} f^s_{ij}(x_{ij})$ so that for each $(i, j) \in A$ $f^s_{ij}(x_{ij}) = f_{ij}(x_{ij})$ if x_{ij} is an integer multiple of s. Let $f^s_{ij}(x_{ij})$ be defined so that it is linear in its argument between succes-

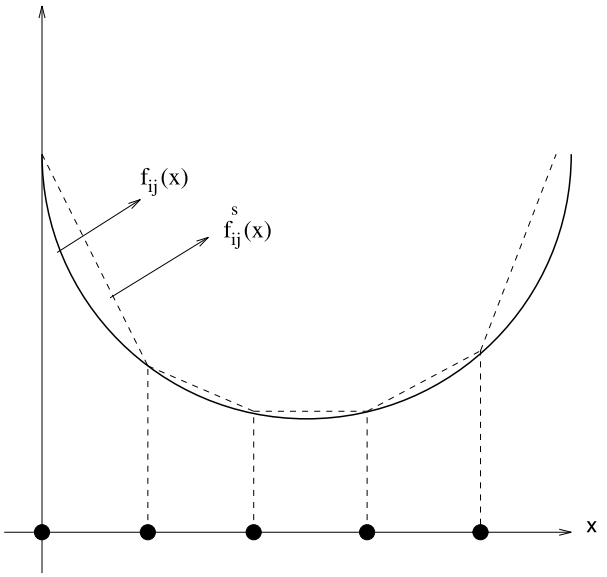

Fig. 1 Piecewise linear approximation

sive integral multiples of s. Thus each function $f_{ij}^s(x_{ij})$ is a piecewise linear approximation of f_{ij} as depicted in Fig. 1. Formally, $f_{ij}^s(s\lfloor\frac{\ell_{ij}}{s}\rfloor + ks) = \sum_{p=1}^{k} \Delta_{ij}^p$ where $\Delta_{ij}^p = f_{ij}(\ell_{ij} + ps) - f_{ij}(\ell_{ij} + (p-1)s)$.

The convex integer network flow, (INF), problem is:

$$\begin{aligned} \text{(INF)} \quad \min \quad & F(\mathbf{x}) \\ \text{s.t.} \quad & T\mathbf{x} = \mathbf{b}, \\ & 0 \le \mathbf{x} \le \mathbf{u}, \\ & \mathbf{x}, \quad \text{integer} \end{aligned}$$

and its continuous (real) relaxation (RNF),

$$\begin{aligned} \text{(RNF)} \quad \min \quad & F(\mathbf{x}) \\ \text{s.t.} \quad & T\mathbf{x} = \mathbf{b}, \\ & 0 \le \mathbf{x} \le \mathbf{u}. \end{aligned}$$

T is an $n \times m$ adjacency matrix of the network $G = (V, A)$, $\mathbf{b}$ is a demand-supply n-vector, and $\mathbf{u}$ the capacity upper bounds vector on the arcs. When separable, the objective function is $F(\mathbf{x}) = \sum_{(i,j)\in A} f_{ij}(x_{ij})$.

The (continuous) problem at the s-scaling phase, for any scaling constant $s \in R_+$, is the scaled problem (RNF-s) obtained by setting $\mathbf{x} = s\mathbf{y}$:

$$\begin{aligned} \text{(RNF-}s\text{)} \quad \min \quad & F^s(s\mathbf{y}) \\ \text{s.t.} \quad & T\mathbf{y} = \frac{\mathbf{b}}{s}, \\ & \mathbf{0} \le \mathbf{y} \le \frac{\mathbf{u}}{s}. \end{aligned}$$

Because of the equality constraints, the integer version (INF-s) (with the requirement that $\mathbf{y}$ is integer) has a feasible solution only if $\mathbf{b}/s$ is integer. Therefore, we choose an equivalent formulation of the scaled problem with *inequality* constraints. The scaled integer problem is then,

$$\begin{aligned}
\text{(INF-}s\text{)} \quad \min \quad & F^s(s\mathbf{y}) \\
\text{s.t.} \quad & T\mathbf{y} \geq \left\lfloor \frac{\mathbf{b}}{s} \right\rfloor, \\
& -T\mathbf{y} \geq -\left\lceil \frac{\mathbf{b}}{s} \right\rceil, \\
& \mathbf{0} \leq \mathbf{y} \leq \frac{\mathbf{u}}{s}, \\
& \mathbf{y} \quad \text{integer}.
\end{aligned}$$

Although a feasible solution to (INF-s) is not necessarily feasible for the original problem, yet the amount of unsatisfied supply (demand) is bounded by $(\lceil \frac{\mathbf{b}}{s} \rceil - \lfloor \frac{\mathbf{b}}{s} \rfloor) \cdot s\mathbf{e} = \sum_{i=1}^{n} (\lceil \frac{b_i}{s} \rceil - \lfloor \frac{b_i}{s} \rfloor) \cdot s \leq ns$ units.[1]

The set of feasible solutions $\{\mathbf{x} \mid T\mathbf{x} = \mathbf{b}, \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}\}$ is bounded, a priori, in a box of length $B = \min\{\| \mathbf{u} \|_\infty, \| \mathbf{b} \|_1\}$ in each dimension as the flow on each edge cannot exceed capacity, or total sum of demands. Denoting $N_{ij} = \lceil \frac{u_{ij} - \ell_{ij}}{s} \rceil$, let each variable x_{ij} be substituted by a sum of N_{ij} variables each bounded between 0 and 1:

$$x_{ij} = s\left\{ \left\lfloor \frac{\ell_{ij}}{s} \right\rfloor + \sum_{k=1}^{N_{ij}} z_{ij}^{(k)} \right\}, \quad 0 \leq z_{ij}^{(k)} \leq 1 \text{ for } k = 1, \ldots, N.$$

In the network the analogous substitution is to replace each arc (i, j) by N_{ij} arcs of capacity 1 each.

The modified objective function for the linear programming formulation of both (LNF-s) and (INF-s) is thus,

$$\min \quad \sum_{(i,j)\in A} f_{ij}^s\left(s\left\lfloor \frac{\ell_{ij}}{s} \right\rfloor\right) + \sum_{(i,j)\in A} \sum_{k=1}^{N_{ij}} \Delta_{ij}^k z_{ij}^{(k)}.$$

Due to the convexity of f_{ij}^ss, the sequence of increments Δ_{ij}^k for $k = 1, \ldots, N_{ij}$ is monotone nondecreasing. This property allows to solve the problem as linear programming without enforcing the integer constraints that $z_{ij}^{(k+1)} > 0$ only if $z_{ij}^{(k)} = 1$, as stated in Lemma 1 below. Let the column of T corresponding to arc (i, j) be denoted by $\mathbf{a}_{ij}$. In the formulation each such column is duplicated N_{ij} times, and in the network each arc is multiplied N_{ij} times where each duplicated arc has the capacity 1. Let T^N be the matrix T in which each column is duplicated N_{ij} times. The constraint set is then,

$$\begin{aligned}
& T^N\mathbf{z} \geq \mathbf{b}', \\
& -T^N\mathbf{z} \geq -\mathbf{b}'
\end{aligned}$$

[1] It is interesting to note that Edmonds and Karp (1972) used such idea of *capacity scaling* for the maximum flow problem that can be formulated as a minimum cost problem with $\mathbf{b} = 0$. This network flow problem readily provides feasible integer solutions as the right hand sides are 0 and thus integers. For the minimum cost flow problem however, feasibility is a concern, which is why we use inequalities in the formulation.

where $b'_p = b_p/s - \sum(\mathbf{a}_{ij})_p \lfloor \ell_{ij}/s \rfloor$ which is the net supply in units of s. The equivalent linear programming formulation of the (LNF-s) problem is then (omitting the constant from the objective function),

$$
\begin{aligned}
(\text{LNF-}s) \quad \min \quad & \sum_{(i,j)\in A} \sum_{k=1}^{N_{ij}} \Delta_{ij}^k z_{ij}^{(k)} \\
\text{s.t.} \quad & T^N \mathbf{z} \geq \mathbf{b}', \\
& -T^N \mathbf{z} \geq -\mathbf{b}', \\
& 0 \leq z_{ij}^{(k)} \leq 1, \quad k = 1, \ldots, N_{ij}, \ (i,j) \in A.
\end{aligned}
$$

Since the early 60's it is well known that such linear programs with monotone increments corresponding to convex functions solve the original piecewise linear convex optimization problem.

Lemma 1 (Dantzig 1963) *Let* $\hat{\mathbf{z}}$ *be an optimal solution to* (LNF-s). *If* f_{ij} *is convex for each* $(i,j) \in A$ *then* $\hat{\mathbf{x}}$ *defined by* $\hat{\mathbf{x}}_{ij} = s\lfloor \frac{\ell_{ij}}{s} \rfloor + \sum_{k=1}^{N_{ij}} \hat{z}_{ij}^{(k)}$, *for all* (i,j), *is an optimal solution to* (RNF-s).

Due to the total unimodularity of the constraint matrix, any optimal solution to the linear program (LNF-s) is also an optimal solution to (INF-s).

It follows from this discussion that solving the integer problem amounts to solving (INF-1) and solving the continuous problem is equivalent to solving (INF-ϵ). The complexity however depends on the number of segments in the piecewise linear approximation, which is exponential. The proximity results in the next section lead to a polynomial algorithm that uses a piecewise linear approximation with only a polynomial number of segments.

There have been several studies that considered how optimizing piecewise linear functions depends on the number of pieces. Sun et al. (1993) observed empirically that solving a transportation problem with a piecewise linear objective function using simplex is not sensitive to the number of segments in that function. Hochbaum and Seshadri (1993) observed a similar result for an implementation of the interior point method. The use of the proximity results of the next section allows however to *guarantee* the polynomiality of each piecewise linear optimization and of the overall procedure.

3.2 The proximity theorem

A proximity theorem is a statement on the distance, in L_∞ norm, between the solution to the scaled problem and the optimal solution to the problem. It is equivalently also a statement on the distance between the optimal solution to the scaled problem with a scaling unit s and the optimal solution to the scaled problem with scaling unit $\frac{s}{2}$. (Note that 2 can be replaced by any other constant.) To see that the first implies the latter note that the distance between the two solutions to the scaled problems is at most their sum of distances from the optimum. On the other hand, if the two scaled problems solutions are close, then their distance to an optimal solution to the problem with $s = 1$ (or ϵ) is at most the sum of the distances between the sequence of scaled problems' solutions.

The essence of the proximity-scaling approach is to solve the scaled problem for a scaling unit that is large enough so that the number of binary variables, or pieces, in the piecewise

linear approximation is polynomially small. The proximity theorem applies to any convex separable minimization problem of the form,

$$\min\{F(\mathbf{x}) \mid A\mathbf{x} = \mathbf{b},\ \mathbf{0} \le \mathbf{x} \le \mathbf{u}\}.$$

The constraints can always be written as equality constraints and the upper bounds of $\mathbf{u}$ can be either explicit or implicit. Let Δ denote the largest subdeterminant of the matrix A and n the number of variables.

Theorem 1 (Hochbaum and Shanthikumar 1990) *Let $\mathbf{x}^s$, be an optimal solution to the problem in the s scaling phase and $\mathbf{x}^{\frac{s}{2}}$ an optimal solution in the $\frac{s}{2}$ scaling phase. Then*

$$\|\mathbf{x}^s - \mathbf{x}^{\frac{s}{2}}\|_\infty \le n\Delta s.$$

A special case of this theorem was proved by Granot and Skorin-Kapov (1990) when $F()$ is a convex separable quadratic function. With Theorem 1 and a judicious choice of the scaling unit, the optimal solution in the s scaling phase bounds an interval in which each variable $x_i^{\frac{s}{2}}$ in the optimal solution for the $\frac{s}{2}$ scaling phase lies. The size of this interval is half the size of the previous interval, thus shrinking the range in the next scaling phase by a factor of 2.

For the convex network flow problem the constraint matrix is totally unimodular and thus $\Delta = 1$.

3.3 A formal description of the algorithm

The proximity-scaling algorithm can be employed whenever there is a valid proximity theorem. For convex network flow the proximity theorem is $\|\mathbf{x}^s - \mathbf{x}^{\frac{s}{2}}\|_\infty \le ms$. We call α the *proximity factor* if $\|\mathbf{x}^s - \mathbf{x}^{\frac{s}{2}}\|_\infty \le \alpha s$.

The algorithm is implemented as follows. The scaling unit is selected initially to be $s = \lceil \frac{U}{4\alpha} \rceil$ for $U = \max_{(i,j)\in A}\{u_{ij} - \ell_{ij}\}$. The interval for variable x_{ij}, $[\ell_{ij}, u_{ij}]$ is thus replaced by up to 4α intervals of length s each.

Proximity-scaling algorithm:

Step 0: Let $s = \lceil \frac{U}{4\alpha} \rceil$.

Step 1: Solve (LNF-s) or (INF-s) with an optimal solution $\mathbf{x}^s$. If $s = 1$ output the solution and stop.

Step 2: Set $\ell_{ij} \leftarrow \max\{\ell_{ij}, x_{ij}^s - \alpha s\}$ and $u_{ij} \leftarrow \min\{u_{ij}, x_{ij}^s + \alpha s\}$, for $(i, j) \in A$.

Step 3: $s \leftarrow \lceil \frac{s}{2} \rceil$. Go to step 1.

4 Proximity-scaling for convex network flow problem

In order to apply the proximity-scaling approach we need to show how to solve the scaled problem (INF-s).

The scaled network flow problem, or any piecewise linear convex cost network flow problem, can be stated as a linear cost network flow problem. To do that one replaces each arc carrying a piecewise linear cost by multiple arcs each with the cost of one of the pieces (or

segments). In our problem, each arc is replaced by $4m$ unit capacity arcs creating a multigraph. Obviously, any polynomial time algorithm for the linear problem is adaptable to solving the piecewise linear problem where the number of arcs includes the multiple arcs. Therefore the complexity depends on the number of grid segments, or alternatively on the scaling constant.

Algorithms dealing with flows on multigraphs have been studied. One such algorithm for the network flow problem with multiple arcs is given in Ahuja et al. (1984). Pinto and Shamir (1994) presented an algorithm that is more efficient than the straightforward approach, yet still depends on the total number of grid segments. Fourer (1988) proposed a practical method for solving a convex piecewise linear program relying on the simplex method but no theoretical complexity guarantees are established.

One way which is particularly efficient for solving the scaled problem in step 1 is an adaptation of the *successive shortest paths method* (due to Jewell 1958; Iri 1960; Busacker and Gowen 1961 with some improvements by Edmonds and Karp 1972). We remark later about potential other approaches for solving the scaled problem. The adaptation we use differs from the original successive shortest paths method (referred to as SSPM) in several aspects.

1. There may be no augmenting path between a node with an excess and any node with a deficit. This does not lead to a termination of the algorithm (due to infeasibility) such as in SSPM. Instead the algorithm proceeds until there are no feasible paths between *any* excess node and *any* deficit node. (Definitions of excess and deficit nodes are given below.)
2. The attempt is only to satisfy excess and deficits that exceed the value s, even though there could be additional excess and deficit nodes.
3. The shortest paths calculation is performed in a multigraph, and thus requires the maintenance of the set of arcs between each pair of nodes sorted by increasing costs. Notice that the sorting of the reduced costs is identical to the sorting of the original costs.
4. The augmentation here is always of 1 unit as it uses the minimum cost arc (which is, like all others, of capacity 1) between each pair of nodes on the path.

For a given solution $\mathbf{x}$ we define the *excess* at j, $\bar{e}_j = \frac{b_i}{s} - Tx_j$. Negative excess is referred to as *deficit*. $G(\mathbf{x})$ is the residual graph with respect to $\mathbf{x}$. r_{ijq} is the residual capacity on the qth arc between i and j which could be 0 or 1. $\boldsymbol{\pi}$ is the vector of dual costs, also known as node potentials. The reduced cost of the qth arc between i and j with cost $c_{ijq} = \Delta_{ij}^q$ is $c_{ijq}^{\pi} = c_{ijq} - \pi_i + \pi_j$.

The procedure Scaled Successive Shortest Paths works with a solution that satisfies capacity constraints, and a dual feasible solution $\boldsymbol{\pi}$. The procedure is called for a scaling unit s once the problem with the scaling unit $2s$ has been solved with a solution $\mathbf{x}^{2s}$, a dual feasible solution $\boldsymbol{\pi}$, and updated upper and lower bounds vectors $\mathbf{L}^{(s)}$, $\mathbf{U}^{(s)}$. The procedure for s is initialized with the solution $\mathbf{x} = \max\{2 \cdot \mathbf{x}^{2s} - s\mathbf{e}, \mathbf{L}^{(s)}\}$. This guarantees that all the reduced costs of residual arcs on pieces of size s are nonnegative and that the dual solution $\boldsymbol{\pi}$ is feasible. This follows from the convexity: if $x_{ij}^{2s} = q$ and c_{ijq} is the cost on the respective interval of size $2s$ that is now split into two intervals of size s, the first of cost $c_{ijq_1} \leq c_{ijq}$ and the second of cost $c_{ijq_2} \geq c_{ijq}$, then $c_{ijq_1} - \pi_i + \pi_j \leq c_{ijq} - \pi_i + \pi_j = 0$ as required. But $c_{ijq_2} \geq c_{ijq}$ and therefore the residual arc of the reverse q_2th arc between i and j may have a negative cost. To avoid that, we subtract s from each entry of $\mathbf{x}^{2s}$ and then the forward direction of this arc is residual and $c_{ijq_2} - \pi_i + \pi_j \geq c_{ijq} - \pi_i + \pi_j = 0$ as required.

In the first call to the procedure the input has $\boldsymbol{\pi} = 0$ and $\mathbf{x} = 0\mathbf{e}$.

Procedure Scaled Successive Shortest Paths:

Input: $\mathbf{L}^{(s)}, \mathbf{U}^{(s)}, \mathbf{x} = \max\{2 \cdot \mathbf{x}^{2s} - s\mathbf{e}, \mathbf{L}^{(s)}\}, s, \pi$.
Step 0: $E_s = \{j \in V \mid \bar{e}_j \geq 1\}$, $D_s = \{j \in V \mid \bar{e}_j \leq -1\}$.
Step 1: If $E_s = \emptyset$ or $D_s = \emptyset$, output $\mathbf{x}$ and stop.
Else, select $k \in E_s$,
Find the shortest paths from k to all nodes in $G(\mathbf{x})$ using the reduced costs c^{π}_{ijq}. Let d_j be the shortest path distance from k to j. If no path exists between k and any node of D_s, set $E_s \leftarrow E_s - \{k\}$ and repeat.
Step 2: Else, let P be a shortest path between k and one $\ell \in D_s$. Update $\mathbf{x}$ by augmenting 1 unit along the arcs of P.
Step 3: Update: $G(\mathbf{x})$, $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} - \mathbf{d}$.
$\bar{e}_k \leftarrow \bar{e}_k - 1$, $\bar{e}_\ell \leftarrow \bar{e}_\ell + 1$.
If $\bar{e}_k < 1$, $E_s \leftarrow E_s - \{k\}$.
If $\bar{e}_\ell > -1$, $D_s \leftarrow D_s - \{\ell\}$.
Go to step 1.

The SSPM algorithm works with a solution that is capacity feasible, i.e. all the capacity upper and lower bounds are satisfied. The solution at each iteration is dual feasible. This holds for the initial solution as discussed above, and the updating $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} - \mathbf{d}$ with $\mathbf{d}$ the vector of shortest paths distances, maintains the nonnegativity of reduced costs.

In the graph processed each arc is duplicated $O(m)$ times. These copies of the arc all have unit capacity and they are sorted in nondecreasing costs. The single source shortest paths can be evaluated in the same running time as if there were single arcs between each pair of nodes since among all the duplicated arcs, there is only one of lowest cost to be considered, and maintaining the arcs sorted is straightforward. The running time required to solve the single source shortest paths problem, using Dijkstra's algorithm is $O(m + n \log n)$, where $|V| = n$ and $|A| = m$.

At each iteration, either the number of nodes with excess is reduced or the flow is augmented. The number of calls to the shortest paths procedure is therefore not exceeding the total excess (or total deficit) in the network. In order to evaluate the total excess at each iteration consider the following. For a given problem there are initially $\frac{1}{2} \parallel \mathbf{b} \parallel_1 = \frac{1}{2} \sum_{i=1}^{n} |b_i|$ units of excess. Each supply value b_i is effectively rounded down to $s\lfloor \frac{b_i}{s} \rfloor$ whereas each demand value (which is a negative number) is rounded down in absolute value, i.e. to $s\lceil \frac{b_i}{s} \rceil$. Once all these demands and supplies $\bar{e}_j$ are satisfied there are up to n unit multiples of s yet unsatisfied – one for each node. Since each scaling iteration is initialized with a vector as low as $2 \cdot \mathbf{x}^{2s} - s\mathbf{e}$ this can add up to m units of excess. Also, because capacity upper bounds in (INF-s) are effectively rounded down, it may be impossible to find a path of capacity 1 in the residual network from an excess node to a deficit node. Each arc can prevent at most one unit of excess from getting canceled against deficit. Hence, applying the scaled successive shortest paths algorithm at an iteration will result in a solution satisfying all but $O(n + m)$ unit multiples of s of supply and demand. Therefore, starting the iteration for scaling unit s with the initial solution $\mathbf{x} = \max\{2 \cdot \mathbf{x}^{2s} - s\mathbf{e}, \mathbf{L}^{(s)}\}$, $\mathbf{x}$ is capacity-feasible and dual feasible, while at most $O(m + n)$ units of excess need to be processed.

The overall complexity of the scaled successive shortest paths algorithm is therefore $O((m + n)(m + n \log n))$. Since there are $\log \frac{B}{m}$ calls to this algorithm, the running time of the proximity-scaling algorithm for the integer convex problem is $O(\log \frac{B}{m}(m + n)(m + n \log n))$, and for the ϵ-accurate solution it is $O(\log \frac{B}{\epsilon}(m + n)(m + n \log n))$.

Other polynomial algorithms As noted above, *any* method that solves the scaled problem can be applied within the proximity-scaling algorithm. Therefore any polynomial time

algorithm for the minimum cost flow problem on the multigraph can be used to generate a polynomial time algorithm for the convex problem.

Minoux (1984, 1986) was first to discover a polynomial algorithm for the integer convex flow problem. Minoux adapted the out-of-kilter method with scaling, so that at each iteration there is a factor reduction in the sum of kilter numbers. The reported running time in Minoux (1986) is $O(\log \|\mathbf{u}\|_\infty \cdot mn^2)$.

Ahuja et al. (1993) introduced another algorithm for the integer convex network flow problem using a capacity scaling algorithm. They again use the solution at one scaling step as the initial solution in the next iteration for the next scaling step. The running time they report, $O(\log \|\mathbf{b}\|_\infty \cdot m(m + n \log n))$, is the same as that of the proximity-scaling algorithm. Interestingly, the complexity of both these algorithms is the same as that of the capacity scaling algorithm applied to the *linear* network flow problem.

All these polynomial algorithms have running times that depend on $\log_2 B$ (recall that $B = \min\{\|\mathbf{u}\|_\infty, \|\mathbf{b}\|_1\}$), which is essentially the length of the right hand sides. Since, as observed for the capacity scaling algorithm, one can achieve algorithms for the convex case with the same running time as the linear case, it seems conceivable that the strongly polynomial algorithms for linear network flow problems could also be adapted to the convex case. This however is impossible as proved in the next section.

An algorithm of different type, by Karzanov and McCormick (1997), for convex cost network flow, is based on the minimum mean cycle canceling and has running time of $O(mn \log n \log(nC))$, where C is the largest cost coefficient. Note that in our complexity model the value of C is the largest cost increment of the function over a unit interval and it is not available with the input, explicitly or implicitly. Therefore a running time which is a function of C cannot be viewed as a function of the input length. Even if the functions are analytic and provided as such, it takes in general a nontrivial amount of running time to evaluate a bound on the value of C by finding the minimum of each of the n convex functions over the relevant interval. Moreover, the exact value of C cannot even be evaluated in polynomial time. For specific objective functions however, where the value of C can be bounded, this algorithm can be faster. This is the case, for instance, for the problem of matrix scaling, Rote and Zachariasen (2007).

With the results in the next section, the algorithms with the same running time as the proximity-scaling algorithm are close to being optimal (with smallest complexity possible). This statement holds in the sense that in order to derive more efficient algorithms for the convex case, there must be more efficient algorithms for the linear case of type that depend on the right hand sides in their complexity. We believe that any such improvement would be extremely challenging.

5 Proximity-scaling for the general allocation problem

The resource allocation problem and its variants are reviewed in details in a comprehensive book by Ibaraki and Katoh (1988). The proximity-scaling procedure by Hochbaum (1994) described here for the resource allocation problem's variants has the lowest complexity among all algorithms for these problems to date. The resource allocation problems are all characterized by being solvable by a greedy algorithm in pseudopolynomial time as described below.

Consider the simple resource allocation problem SRA,

$$\begin{aligned} \text{(SRA)} \quad \max \quad & \sum_{j=1}^{n} f_j(x_j) \\ \text{s.t.} \quad & \sum_j x_j \geq B, \\ & x_j \geq 0, \quad \text{integer}, \ j = 1, \ldots, n. \end{aligned}$$

The objective function is concave separable, so $f_j()$ are all concave. This allocation problem is of algorithmic interest in that it is solvable optimally by a greedy algorithm. The constraint is satisfied with equality if all functions $f_j()$ are monotone nonincreasing. In fact, if any of these functions has maximum at $x_j = \ell_j > 0$ then we can always replace the lower bound of 0 for x_j by ℓ_j (or the last value where the function's increment is still nonnegative). If $\sum_j \ell_j \geq B$ then we found the optimal solution $x_j = \ell_j$. Therefore we will assume without loss of generality that the functions are in the range where they are monotone nonincreasing and the constraint is an equality constraint $\sum_j x_j = B$.

An important concept used by the greedy algorithm is that of an *increment*. Let $\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j)$ be the increment of the function $f_j()$ at x_j. The greedy algorithm picks one largest increment at a time until $B - \sum_j \ell_j$ increments have been selected. The complexity of this algorithm is of course not polynomial, but rather depends on the parameter B and is thus pseudo-polynomial.

The most general case of the allocation problem involves separable concave maximization over polymatroidal constraints: Given a *submodular* rank function $r : 2^E \to R$, for $E = \{1, \ldots, n\}$, i.e. $r(\phi) = 0$ and for all $A, B \subset E$,

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B).$$

The *polymatroid* defined by the rank function r, is the polytope $\{\mathbf{x} \mid \sum_{j \in A} x_j \leq r(A), A \subseteq E\}$. We call the system of inequalities $\{\sum_{j \in A} x_j \leq r(A), A \subseteq E\}$, the *polymatroidal constraints*. The general allocation problem, GAP, is

$$\begin{aligned} \text{(GAP)} \quad \max \quad & \sum_{j=1}^{n} f_j(x_j) \\ \text{s.t.} \quad & \sum_j x_j = B, \\ & \sum_{j \in A} x_j \leq r(A), \ A \subset E, \\ & x_j \geq \ell_j, \quad \text{integer}, \ j = 1, \ldots, n. \end{aligned}$$

The problem GAP is also solvable by a greedy algorithm:

Procedure greedy:

Input: $\{\ell_j\}_{j=1}^n$, $r()$, E.
Step 0: $x_j = \ell_j$, $j = 1, \ldots, n$, $B \leftarrow B - \sum_j \ell_j$.
Step 1: Find i such that $\Delta_i(x_i) = \max_{j \in E} \Delta_j(x_j)$.
Step 2: {feasibility check} If $\mathbf{x} + \mathbf{e}^i$ is infeasible then $E \leftarrow E \setminus \{i\}$
else, $x_i \leftarrow x_i + 1$ and $B \leftarrow B - 1$.

Step 3: If $B = 0$, output $\mathbf{x}$ and stop. If $E = \emptyset$, output "no feasible solution" and stop.
Go to step 1.

Obviously this algorithm is pseudopolynomial. Consider now a scaled form of the greedy defined on the problem scaled on units of length s. Let the solution delivered by greedy(s) be denoted by x^s. The procedure here benefits from a correction of an error in step 2 of the procedure in Hochbaum (1994) that was noted and proved by Moriguchi and Shioura (2004). (The error was to set the value of δ_i to be equal to 1 if increment of 1 is feasible but increment of s is infeasible.)

Procedure greedy(s):

Step 0: $\delta = \mathbf{0}$, $x_j = \ell_j$, $j = 1, \ldots, n$, $B \leftarrow B - \sum_j \ell_j$.
Step 1: Find i such that $\Delta_i(x_i) = \max_{j \in E} \Delta_j(x_j)$.
Step 2: {feasibility check} If $\mathbf{x} + \mathbf{e}^i$ is infeasible then $E \leftarrow E \setminus \{i\}$, and $\delta_i = s$.
Else, if $\mathbf{x} + s\mathbf{e}^i$ is infeasible then let $E \leftarrow E \setminus \{i\}$, and let $\delta_i < s$ be largest so that $\mathbf{x} + \delta_i \mathbf{e}^i$ is feasible. Set $x_i \leftarrow x_i + \delta_i$, and $B \leftarrow B - \delta_i$.
Else $x_i \leftarrow x_i + s$ and $B \leftarrow B - s$.
Step 3: If $B = 0$, output $\mathbf{x}^s = \mathbf{x}$, δ, and stop. If $E = \emptyset$, output "no feasible solution" and stop.
Else go to step 1.

We now have the *proximity theorem* for GAP:

Theorem 2 Hoc94 *If there is a feasible solution to GAP then there exists an optimal solution* $\mathbf{x}^*$ *such that* $\mathbf{x}^* > \mathbf{x}^s - \delta \geq \mathbf{x}^s - s\mathbf{e}$.

Based on this proximity theorem we have the following proximity scaling algorithm solving GAP:

Procedure GAP:

Step 0: Let $s = \lceil B/2n \rceil$.
Step 1: If $s = 1$ call *greedy*. Output "$\mathbf{x}^* = \mathbf{x}$ is an optimal solution", stop. Else, continue.
Step 2: Call *greedy(s)*. Let the output be $\mathbf{x}^s$.
Set $\ell_j \leftarrow \max\{x_j^s - \delta_j, \ell_j\}$ for $j = 1, \ldots, n$.
Set $s \leftarrow \lceil s/2 \rceil$.
Go to step 1.
end

This algorithm is valid and its complexity is $O(n(\log n + F)\log(B/n))$ where F is the complexity of determining δ_i – the tightest slack to infeasibility. Furthermore, this proximity-scaling algorithm leads to the fastest algorithms known for all special cases of the general allocation problem (Hochbaum 1994). The complexity expressions of the algorithm for the different cases are:

1. For the simple resource allocation problem SRA, $O(n \log \frac{B}{n})$. This matches the complexity shown earlier by Frederickson and Johnson Frederickson and Johnson (1982) using a different technique.
2. For the generalized upper bounds resource allocation problem, GUB, $O(n \log \frac{B}{n})$.
3. For the *nested* problem, $O(n \log n \log \frac{B}{n})$.

4. For the *tree constrained* problem, $O(n \log n \log \frac{B}{n})$.

The complexity bounds for SRA and GUB are also shown to be best possible in the comparison model.

6 The nonlinear knapsack problem

The Nonlinear knapsack problem (NLK) is a generalization of the well known integer knapsack problem which maximizes a linear objective function representing utilities associated with choosing items (the number of units of item j is represented by the variable x_j) subject to a "packing" constraint: $\max\{\sum_{j=1}^{n} p_j x_j \mid \sum_{j=1}^{n} a_j x_j \le B,\ u_j \ge x_j \ge 0,\ \text{integer for } j = 1, \ldots, n\}$. In its general form the nonlinear knapsack problem has the objective separable concave and the packing constraint separable convex:

$$\begin{aligned} \text{(NLK)} \quad \max \quad & \sum_{j=1}^{n} f_j(x_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} g_j(x_j) \le B, \\ & 0 \le x_j \le u_j, \quad \text{integer}, \ j = 1, \ldots, n. \end{aligned}$$

The functions f_j are assumed concave and nondecreasing, and the functions g_j are assumed convex and nondecreasing. Without loss of generality B and u_j are integers.

The results sketched here were proved in Hochbaum (1995) based on a proximity theorem and an analogy of NLK to SRA. The continuous problem is shown to be solvable with an ϵ-accurate solution in time $O(n \log B/\epsilon)$. This running time is impossible to improve as it is equal to the running time for solving the continuous SRA problem, which is a simple special case of the nonlinear knapsack problem.

A piecewise linear approximation of the functions f_j and the functions g_j is used to convert the nonlinear knapsack problem (NLK) into a 0/1 knapsack problem. The piecewise linear approximation on the integer grid for the objective of NLK is achieved by replacing each variable x_j by the sum of binary variables $\sum_{j=1}^{u_j} x_{ij}$, and letting $p_{ij} = f_j(i) - f_j(i-1)$, and $a_{ij} = g_j(i) - g_j(i-1)$:

$$\begin{aligned} \text{(PLK)} \quad \max \quad & \sum_{j=1}^{n} \sum_{j=1}^{u_j} p_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} \sum_{j=1}^{u_j} a_{ij} x_{ij} \le B, \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots u_j, \ j = 1, \ldots, n. \end{aligned}$$

It is easy to see that the concavity of f_j and the convexity of g_j guarantee that $x_{ij} > 0$ only if $x_{i,j-1} = 1$. It follows that when a_{ij} and p_{ij} are integers, then techniques that are used for the 0/1 knapsack problem are applicable here as well.

The problem PLK is a 0/1 knapsack problem $\max\{\sum_{j=1}^{N} p_j x_j \mid \sum_{j=1}^{N} a_j x_j \le B, 1 \ge x_j \ge 0 \text{ integer}, j = 1, \ldots, N\}$. The complexity of solving the 0/1 knapsack problem with a well known dynamic programming algorithm is $O(N \cdot \min\{B, P^*\})$ for P^* denoting the

optimal solution value. For this dynamic programming to work, it is necessary that both the objective function and constraint coefficients are integral. Otherwise, the dynamic programming algorithm runs in $O(NB)$ operations if only the constraint coefficients (the weights) are integral. It runs in time $O(NP^*)$ if only the objective function coefficients are integral.

Thus, if the functions g_j map integers to integers or the functions f_j map integers to integers, then NLK is solvable in $O(B\sum_{j=1}^{n} u_j)$ steps, or $O(P^*\sum_{j=1}^{n} u_j)$ steps, respectively. This is the same complexity as that of the corresponding linear knapsack problem, but unlike the term n, the term $\sum_{j=1}^{n} u_j)$ is not polynomial.

6.1 Solving the continuous nonlinear knapsack problem

The key idea is to use a representation of a scaled piecewise linear NLK problem as an SRA problem in order to generate a polynomial time algorithm.

$$\begin{aligned} \text{(NLK-allocation)} \quad \max \quad & \sum_{j=1}^{n}\sum_{j=1}^{u_j} \frac{p_{ij}}{a_{ij}} y_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n}\sum_{j=1}^{u_j} y_{ij} \le B, \\ & y_{ij} \quad \text{integer},\ i = 1,\ldots,u_j,\ j = 1,\ldots,n. \end{aligned}$$

Let the vector $\mathbf{y}^*$ be the optimal solution to (NLK-allocation). Let $\bar{y}_{ij} = \frac{1}{aij} y^*_{ij}$ for all i, j. The allocation proximity theorem (see Sect. 5) implies that $\mathbf{x}^* \ge \mathbf{x}^s - s \cdot \mathbf{e}$. But since x^s also satisfies $\mathbf{x}^s \cdot \mathbf{e} = B$, then $\|\mathbf{x}^* - \mathbf{x}^s\|_\infty \le ns$ and thus $\|\mathbf{x}^* - \bar{\mathbf{y}}\|_\infty \le n \max_{ij} \frac{1}{a_{ij}}$. Consequently, there is a proximity between the optimal solution to NLK and the optimal solution to NLK-allocation and it is sufficient to solve the latter.

In order to obtain ϵ-accuracy we modify the transformation of variables to $x_{ij} = \frac{y_{ij}}{s_{ij}}$, where $s_{ij} = a_{ij}\lceil \frac{n}{\epsilon} \rceil$ and duplicate each entry (i, j) s_{ij} times. Although this increases the size of the arrays, it does not cause an increase in the running time required to solve the allocation problem (NLK-allocation) as that depends only on the number of arrays and the right hand side value. The right hand side is also scaled so that all coefficients are integer: $\bar{B} = B\lceil \frac{n}{\epsilon} \rceil$. Consequently, the running time is $O(n \log \frac{\bar{B}}{n}) = O(n \log \frac{B}{\epsilon})$. The lower bound for the allocation problem implies that this complexity is impossible to improve.

6.2 Fully polynomial approximation scheme for nonlinear knapsack

The fully polynomial time approximation scheme (FPTAS) for NLK builds on the link of the problem to the allocation problem. We sketch it briefly here. For the full details the reader is referred to Hochbaum (1995). The scheme mimics that of Lawler's (Lawler 1979) that uses the dynamic programming algorithm that solves the problem in $O(nP^*)$. Lawler's approximation scheme's complexity is not polynomial as it include the factor $\sum_{j=1}^{n} u_j$ in the running time.

The objective function coefficients are scaled thus reducing the running time of the algorithm to depend on the new scaled value of the optimal solution. In addition, for a carefully chosen scaling value the objective function of the scaled problem is close to that of the original problem. Basically, this procedure implements efficiently the steps of the linear knapsack problem's FPTAS for NLK:

1. Find the value and the set of elements corresponding to $P_0 = \max\{\max_j p_j, \sum_{j=1}^{\bar{j}} p_j\}$, for $\bar{j}$ the largest index so that when the variables in the 0/1 knapsack are arranged in nonincreasing ratio, then $\sum_{j=1}^{\bar{j}} a_j \leq B$.
2. Find the "large" items that are candidates for inclusion in the optimal solution. "Large" items are those with profit coefficients $p_j \geq \frac{1}{2}\epsilon P_0$. This is done using the SRA algorithm.
3. Solve the scaled problem for the "large" items, using dynamic programming.
4. Find the largest ratio "small" items that can be packed in the remaining capacity of the knapsack.

The union of the set of items—the large ones found in step 3 and the small ones found in step 4—form the approximate solution to the problem. We skip the proof that the approximation factor provided is indeed ϵ.

The running time of this ϵ-approximation scheme is $\tilde{O}(1/\epsilon^2)(n + 1/\epsilon^2))$. (The $\tilde{O}$ notation indicates the omission of polylog terms.)

7 Convex dual of network flow

The dual of the minimum cost network flow problem on a graph $G = (V, A)$ is characterized by constraints of the type $x_i - x_j \leq c_{ij} + z_{ij}$, for z_{ij} nonnegative and each arc $(i, j) \in A$. The objective function is of the type $\min \sum_{j\in V} f_j(x_j) + \sum_{(i,j)\in A} g_{ij}(z_{ij})$. The problem has numerous applications including the dial-a-ride transit problem and the time-cost trade-off in project management. For these and additional applications the reader is referred to Ahuja et al. (2003). This problem has been addressed with a proximity-scaling algorithm in Ahuja et al. (2004).

The interesting feature about that algorithm is that the scaled piecewise linear version of the problem is a minimum cut problem on an appropriately defined graph. The proximity-scaling algorithm calls $\log U$ times for a minimum cut procedure where $U \leq n \max_{(i,j)\in A} |c_{ij}|$ is the largest interval for a variable x_j. The minimum cut procedure is applied to a graph of size that is square the size of the original graph. This algorithm's complexity is worse than that of another algorithm by Ahuja et al. (2003) which uses successive shortest paths to solve the problem and has complexity $O(mn \log \frac{n^2}{m} \log U)$. We do not describe this algorithm here as the technique is specialized and does not appear to have implications for other convex optimization problems.

8 Inverse shortest paths and the use of a projected proximity theorem in a proximity-scaling setup

In the inverse paths problem there is a given graph with arc weights and given "shortest paths" distances from a source to a collection of nodes. The goal is to modify the given arc weights as little as possible so that the prescribed "shortest paths" routes are indeed the shortest paths. The cost of deviation of the arc weights from their given values is a convex function of the amount of the deviation. Previously known polynomial time algorithms for the inverse shortest paths problem were given only for the case of L_1 norm and for a single path (see e.g. Ahuja and Orlin 2001b).

Inverse shortest paths problems have applications in contexts of pricing of communication networks, where in order to be competitive the prices offered for linking services should

be kept low, or in deducing the imputed value of transportation links. Obtaining geophysical information from the detection of the traversal of seismic waves is the most common application.

Some of the limitations of solving this class of problems to date have to do with the characterization of shortest paths and the problem formulation. Burton and Toint (1992) described a formulation with exponential number of constraints enumerating all paths in the graph and restricting their length to be less than the adjusted length of the route which is to be shortest. Hochbaum (2002) devised compact alternative formulations that lead to better and more efficient solution techniques that are not restricted to linear or quadratic penalty functions.

We briefly review here the problems and techniques for the single source shortest paths with prescribed paths problem and the p sources shortest paths with prescribed paths. Another interesting variant of the problem, not reviewed here for lack of space, is the "correlated costs" shortest paths previously discussed by Burton and Toint (1994). For additional details see Hochbaum (2002).

8.1 The single source multisink problem with prescribed shortest paths distances

In this problem we have a source node 1 and a set of destinations $V' \subset V$ for which the shortest paths routes in the form of a shortest paths tree are known. Formally we are given a graph $G = (V, A)$ with a source node 1, estimated edge distances c_{ij} for $(i, j) \in A$, and the observed shortest paths tree, $T' \subset A$. The inverse shortest paths problem (ISP) is to modify the edge distances so that the shortest paths with respect to the modified distances are as prescribed by the tree T'. Let the penalty functions for modifying the edge distance on edge (i, j) from c_{ij} to x_{ij} be the convex functions $f_{ij}(x_{ij} - c_{ij})$, and let the variables t_j be the shortest paths labels from node 1 to node $j \in V$.

Let $D = n \max_{(i,j)\in A} |c_{ij}|$. If the edge distances are all positive then the bound constraints are modified to, $0 \le t_i \le D$.

$$
\begin{aligned}
(\mathrm{ISP}_1) \quad \text{Min} \quad & \sum_{(i,j)\in A} f_{ij}(x_{ij} - c_{ij}) \\
\text{subject to} \quad & t_j - t_i \le x_{ij}, \quad \forall (i, j) \in A \setminus T', \\
& t_j - t_i = x_{ij}, \quad \forall (i, j) \in T', \\
& t_1 = 0, \\
& -D \le t_i \le D, \quad \forall i \in V.
\end{aligned}
$$

The problem ISP_1 is a convex dual of the minimum cost network flow problem. The running time of the algorithm of Ahuja et al. (2003) for a graph of n nodes and m arcs is $O(mn \log \frac{n^2}{m} \log(nD))$.

8.2 The k paths problem

In terms of complexity the problem on multiple sources and destinations pairs is more involved than that of the single source problem. The problem is defined on an input of k paths with multiple sources $u_1, \ldots, u_k$ and multiple sinks (or destinations) $v_1, \ldots, v_k$, $P_q = [u_q, \ldots, v_q]$ for $q = 1 \ldots, k$. Also given are the prior distance estimates on each arc c_{ij}. Let $t_i^{(q)}$ be variables denoting the shortest paths labels from source u_q to node i. The

problem is to determine the values of the modified arc distances x_{ij} that are optimal for the problem:

$$
\begin{aligned}
(\mathrm{ISP}_2)\quad \text{Min}\quad & \sum_{(i,j)\in A} f_{ij}(x_{ij}-c_{ij}) \\
\text{subject to}\quad & t_j^{(q)} - t_i^{(q)} \le x_{ij}, \quad q=1,\ldots,k,\ \forall (i,j)\in A, \\
& t_j^{(q)} - t_i^{(q)} = x_{ij}, \quad q=1,\ldots,k,\ \forall (i,j)\in P_q, \\
& t_{u_q}^{(q)} = 0, \quad q=1,\ldots,k.
\end{aligned}
$$

The constraint matrix of ISP_2 is not totally unimodular. This problem is the convex dual of the multicommodity flow problem, *multicut*. The multicut problem is NP-hard to solve in integers, even for a linear objective function, so this seemingly eliminates the possibility of using combinatorial techniques for solving the problem.

Nevertheless it is possible to solve the problem as a linear problem. The idea is to use the proximity result in Theorem 3 below with the proximity-scaling algorithm in order to reduce this convex problem to a linear programming counterpart.

Let the scaled problem called (s-ISP) be defined on the variables $x_j^{[s]} = \frac{x_j}{s}$, and $z_{ij}^{[s]} = \frac{z_{ij}}{s}$. Let the functions $w_j^{[s]}()$ and $f_{ij}^{[s]}()$ be piecewise linear convex functions that coincide with the convex functions $w_j()$ and $f_{ij}()$ at breakpoints that are s units apart.

Adding the implied implicit bounds on the values of the distances, the scaled problem is

$$
\begin{aligned}
(\text{s-ISP}_2)\quad \text{Min}\quad & \sum_{(i,j)\in A} f_{ij}^{[s]}(s x_{ij}-c_{ij}) \\
\text{subject to}\quad & t_j^{(q)} - t_i^{(q)} \le x_{ij}, \quad q=1,\ldots,k,\ \forall (i,j)\in A, \\
& t_j^{(q)} - t_i^{(q)} = x_{ij}, \quad q=1,\ldots,k,\ \forall (i,j)\in P_q, \\
& t_{u_q}^{(q)} = 0, \quad q=1,\ldots,k, \\
& -\tfrac{D}{s} \le t_i^{(q)} \le \tfrac{D}{s}, \quad \forall i\in V,\ q=1,\ldots,k.
\end{aligned}
$$

For $\mathbf{x}'$ a scaled optimal solution to s-IPS let $\mathbf{x}^s = s\mathbf{x}'$ be the optimal solution vector which is feasible for ISP. A corollary of Theorem 1 is that for a problem in N variables with a largest subdeterminant Δ the distance between the optimal solution $\mathbf{x}^*$ to the problem LP and the optimal solution $\mathbf{x}^s$ to the scaled problem LP-s is

$$\|\mathbf{x}^* - \mathbf{x}^s\|_\infty \le 2Ns\Delta.$$

In our case the number of variables is $O(m)$ but the size of the largest subdeterminant can be exponentially large in the size of the matrix and the size of the coefficients α_k for the multiple paths problem. A proximity theorem 3 addresses this issue by restricting the proximity to a portion of the variables only. Namely, the proximity is of the form $\|\mathbf{t}^* - \mathbf{t}^s\|_\infty \le ns$. More precisely, let the solution vector be the vector $(\mathbf{t},\mathbf{x})$ where $\mathbf{t}\in R^n$ and $\mathbf{x}\in R^m$. We let the problem s-ISP be the problem 1-ISP where the scaling unit is 1.

Theorem 3 (Hochbaum 2002)

(i) *For each optimal solution* $(\mathbf{t},\mathbf{x})$ *for* ISP, *there exists an optimal solution* $(\mathbf{s}^*,\mathbf{x}^*)$ *for* 1-ISP *such that* $\|\mathbf{t}-\mathbf{s}^*\|_\infty \le n$.

(ii) *For each optimal solution* $(\mathbf{s}, \mathbf{x}^s)$ *for* 1-ISP, *there exists an optimal solution* $(\mathbf{t}^*, \mathbf{x}')$ *for* ISP *such that* $\|\mathbf{t}^* - \mathbf{s}\|_\infty \le n$.

The piecewise linear problem s-ISP$_2$ can be solved by linear programming with each variable replaced by $4s$ variables, one for each segment of length $\frac{2D}{s}$. The number of variables in each linear programming problem (s-ISP$_2$) is then four times the number of variables in the problem of the previous scaling iteration.

Procedure inverse paths:

Step 0: Let $s = \frac{U}{4}$.
Step 1: Solve (s-ISP$_2$), with an optimal solution $\mathbf{x}^s$. If $s = 1$ output the solution and stop.
Step 2: Set $\ell_j \leftarrow \max\{\ell_j, x_j^s - s\}$ and $u_j \leftarrow \min\{u_j, x_j^s + s\}$, for $j = 1, \ldots, n$.
Step 3: $s \leftarrow \frac{s}{2}$. Go to step 1.

Here procedure inverse paths executes $O(\log D)$ calls to the linear programming problem (s-ISP$_2$).

9 Threshold theorem based algorithm for convex closure and convex s-excess

A threshold theorem is a particularly strong form of proximity. In a threshold theorem we replace the convex objective by a linear objective where each coefficient is the derivative (or subgradient) of the respective function at some point α. When a threshold theorem holds, we can conclude from the optimal solution to this linear problem that the value of some of the variables at the optimum is greater than α whereas the others are smaller than or equal to α. We illustrate this concept for two problems, the convex cost closure problem and the convex s-excess problem. These problems are characterized by constraints of the form $x_i \ge x_j$ and $x_i - x_j \le z_{ij}$, respectively.

9.1 The convex cost closure problem

A common problem in statistical estimation is that observations do not satisfy preset ranking order requirements. The challenge is to find an adjustment of the observations that fits the ranking order constraints and minimizes the total deviation penalty. Many aspects of this problem as well as numerous applications are studied in (Barlow et al. 1972). The deviation penalty is a convex function of the fitted values. This application motivated the introduction of the convex cost closure problem in Hochbaum and Queyranne (2003).

The convex cost closure problem (CCC) is defined formally on a directed graph $G = (V, A)$ with convex functions $f_j()$ associated with each node $j \in V$. The formulation of the convex cost closure problem is then:

$$\begin{aligned} \text{(CCC)} \quad \min \quad & \sum_{j \in V} f_j(x_j) \\ \text{subject to} \quad & x_i - x_j \ge 0, \quad \forall (i,j) \in A, \\ & \ell_j \le x_j \le u_j, \quad \text{integer}, \ j \in V. \end{aligned}$$

This problem generalizes the (linear) closure problem which is (CCC) with binary variables, that is $\ell_j = 0$ and $u_j = 1$. The closure problem is known to be equivalent to solving a minimum s, t-cut problem in a related graph. This was first noted explicitly by Picard (1976).

The threshold theorem by Hochbaum and Queyranne (2003) reduces the convex problem to its binary counterpart – the minimum closure problem. To sketch the main idea of the theorem we first note that one can extend all the functions $f_i()$ so that they are convex in the range $[\ell, u]$ for $\ell = \min_i \ell_i$, $u = \max_i u_i$. Let α be a scalar and w_i be the derivative or subgradient of f_i at α, $w_i = f_i'(\alpha) = f_i(\alpha+1) - f_i(\alpha)$. Let $G_\alpha = (V, A)$ be a closure graph with node weights w_i. The threshold theorem states:

Theorem 4 (Hochbaum and Queyranne 2003) *Let the optimal closure in G_α be S_α. Then the optimal values of the variables for the convex problem x_j^* satisfy $x_j^* > \alpha$ if $j \in S_\alpha$, and $x_j^* \leq \alpha$ otherwise.*

By repeated applications of the minimum closure algorithm on the graph G_α for a range of values of α in $[\ell, u]$ we obtain a partition of the set of variables and of the interval $[\ell, u]$ into up to n subsets and subintervals where each subinterval contains the optimal value of one subset of variables. It is further shown in Hochbaum and Queyranne (2003) that this partition can be achieved with a parametric minimum cut procedure where α is the parameter.

The procedure used to solve the parametric minimum cut problem is a generalization of a procedure devised by Gallo et al. (1989) for linear functions of the parameter, which are based on the push-relabel algorithm of Goldberg and Tarjan (1988). The generalization for any monotone functions is described in Hochbaum (2003) and in Hochbaum (1998) for both the push-relabel algorithm and the pseudoflow algorithm. The algorithm requires at each iteration finding the integer minima of the convex functions which is accomplished with binary search in $O(n \log U)$ steps. The run time of the procedure solving the convex closure problem is shown to be $O(mn \log \frac{n^2}{m} + n \log U)$ which is the sum of the complexities of a (single) minimum s, t cut procedure and the minimization of n convex functions in bounded intervals of length up to U.

The convex cost closure problem generalizes the minimum cut problem (when the functions are linear), and it is at least as hard as the minimization of n convex functions over bounded intervals (when there are no constraints other than upper/lower bounds). Hence the run time cannot be improved unless the respective run times of the minimum cut problem and minimizing convex functions can be improved.

9.2 The minimum s-excess problem

The s-excess problem is a variant of the maximum/minimum closure problem with a *relaxation* of the closure requirement: Nodes that are successors of other nodes in S (i.e. that have arcs originating from a node of S to these nodes) may be excluded from the set, but at a penalty that is equal to the capacity of those arcs. In a closure graph these arcs are of infinite capacity. For the s-excess problem the arcs have finite capacities representing the penalties for violating the closure requirement.

The minimum s-excess problem is defined on a directed graph $G = (V, A)$, with node weights (positive or negative) w_i for all $i \in V$, and nonnegative arc weights u_{ij} for all $(i, j) \in A$. The objective is to find a subset of nodes $S \subseteq V$ such that $\sum_{i \in S} w_i + \sum_{i \in S, j \in \bar{S}} u_{ij}$ is minimum. (The maximum s-excess problem is to maximize $\sum_{i \in S} w_i - \sum_{i \in S, j \in \bar{S}} u_{ij}$.)

A generalized form of Picard's theorem showing that the closure problem is equivalent to the minimum cut problem has been proved for the s-excess problem in Hochbaum (1998). The idea there was to construct a graph as for the closure problem except that the arc capacities not adjacent to source and sink for (i, j) in A are the respective weights u_{ij}. The sink

set of a minimum cut in the graph created was shown to be the minimum s-excess set. The interested reader is referred to Hochbaum (1998) for details.

The s-excess problem has appeared in several forms in the literature. One is the boolean quadratic minimization problem with all the quadratic terms having positive coefficients. This can be shown to be a restatement of the s-excess problem. Another, more closely related is the feasibility condition of Gale (1957) for a network with supplies and demands, or Hoffman's (Hoffman 1960) for a network with lower and upper bounds. Verifying feasibility is equivalent to ensuring that the maximum s-excess is zero, in a graph with node weights equal to the respective supplies and demands with opposite signs – if the s-excess is positive, then there is no feasible flow satisfying the supply and demand balance requirements. This problem appeared also under the names *maximum blocking cut* or *maximum surplus cut* in Radzik (1993).

9.3 The convex s-excess problem

The convex s-excess problem is a generalization of the s-excess problem with node weights $f_j()$ that are convex functions.

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

This problem was studied in the context of the application of *image segmentation* by Hochbaum (2001). In the problem of image segmentation a transmitted image is degraded by noise. The assumption is that a "correct" image tends to have areas of uniform color. The goal is to reset the values of the colors of the pixels so as to minimize the penalty for the deviation from the observed colors, and furthermore, so that the discontinuity in terms of separation of colors between adjacent pixels is as small as possible. Thus the aim is to modify the given color values as little as possible while penalizing changes in color between neighboring pixels. The penalty function has two components: the deviation cost that accounts for modifying the color assignment of each pixel, and the separation cost that penalizes pairwise discontinuities in color assignment for each pair of neighboring pixels.

Representing the image segmentation problem as a graph problem we let the pixels be nodes in a graph and the pairwise neighborhood relation be indicated by edges between neighboring pixels. Each pairwise adjacency relation $\{i, j\}$ is replaced by a pair of two opposing arcs (i, j) and (j, i) each carrying a capacity representing the penalty function for the case that the color of j is greater than the color of i and vice versa. The set of directed arcs representing the adjacency (or neighborhood) relation is denoted by A. We denote the set of neighbors of i, or those nodes that have pairwise relation with i, by $N(i)$. Thus the problem is defined on a graph $G = (V, A)$. Each node j has the observed color value g_j associated with it. The problem is to assign an integer value x_j, selected from a spectrum of K colors, to each node j so as to minimize the penalty function.

Let the K color shades be a set of ordered values $\mathcal{L} = \{q_1, q_2, \ldots, q_K\}$. Denote the assignment of a color q_p to pixel j by setting the variable $x_j = p$. Each pixel j is permitted to be assigned any color in a specified range $\{q_{\ell_j}, \ldots, q_{u_j}\}$. For $G()$ the *deviation cost* function and $F()$ the *separation cost* function the problem is,

$$\min_{u_i \geq x_i \geq \ell_i} \quad \sum_{i \in V} G_i(g_i, x_i) + \sum_{i \in V} \sum_{j \in N(i)} F_{ij}(x_i - x_j).$$

This formulation is equivalent to the following constrained optimization problem, referred to as (IS) (acronym for Image Segmentation):

$$\begin{aligned} \text{(IS)} \quad \min \quad & \sum_{j \in V} G_j(g_j, x_j) + \sum_{(i,j) \in A} F_{ij}(z_{ij}) \\ \text{subject to} \quad & x_i - x_j \leq z_{ij}, \quad \text{for } (i,j) \in A, \\ & u_j \geq x_j \geq \ell_j, \quad j = 1, \ldots, n, \\ & z_{ij} \geq 0, \quad (i,j) \in A. \end{aligned}$$

The case where the functions $F_{ij}()$ are *concave* is easily shown to be NP-hard. To see that consider a reduction from maximum cut: Given an undirected graph $G = (V, E)$ find a cut so the number of edges across the cut is a maximum. Let the values of x_i be 0 or 1, and the objective be $\min \sum_{(i,j) \in E} -|x_i - x_j|$. This is equivalent to $-[\max_{(i,j) \in E} |x_i - x_j|]$. The partition of V into $V_0 = \{i \in V \mid x_i = 0\}$ and $V_1 = \{i \in V \mid x_i = 1\}$ is therefore a maximum cut.

If the functions $F_{ij}()$ are convex and $G_j()$ are convex then the problem becomes an instance of the dual of minimum cost network flow which is solved in polynomial time as sketched in Sect. 7. Even if $G_j()$ are nonconvex, but $F_{ij}()$ are convex, the problem is solved in time that is polynomial in $U = \max_j \{u_j - \ell_j\}$ as show in Ahuja et al. (2004).

The constraints of (IS) have several interesting properties. Firstly, the coefficients of the constraints form a totally unimodular matrix. Secondly, the set of constraints are those of the linear programming dual of the minimum cost network flow. For the dual of the minimum cost network flow problem, a generic constraint is of the type

$$x_i - x_j \leq c_{ij} + z_{ij}.$$

A threshold theorem for the convex s-excess problem generalizing the one in Hochbaum and Queyranne (2003) was proved in Hochbaum (2001). The essence of the theorem is to reduce the convex s-excess problem to the s-excess problem on binary variables which is equivalent to the ordinary minimum s, t-cut problem (Hochbaum 1998). We construct for any α a graph G_α where the weight of node j is the scalar $w_j = f_j(\alpha + 1) - f_j(\alpha)$ – the subgradient or derivative of f_j at α. The minimum s-excess problem defined on that graph with the objective function $\min \sum_{j \in V} w_j x_j + \sum_{(i,j) \in A} u_{ij} z_{ij}$ is solved as a minimum cut problem. If there are multiple optimal solutions we pick the one where the s-excess set is maximal (i.e. not contained in any other optimal set) and thus unique. The uniqueness follows from the properties of the minimum cut.

Theorem 5 (Hochbaum 2001) *Let S^* be the maximal minimum s-excess set in the graph G_α. Then there is an optimal solution $\mathbf{x}^*$ to the corresponding convex s-excess problem satisfying $x_i^* \geq \alpha$ if $i \in S^*$ and $x_i^* < \alpha$ if $i \notin \bar{S}^*$.*

Let the (IS) problem involve n pixels (variables) and m adjacency relations (arcs). Let $T(n, m)$ be the complexity of solving the minimum s, t cut problem on a graph with n nodes and m arcs. The algorithm based on the threshold theorem solves the problem for

$G(\,)$ convex functions and $F(\,)$ linear functions in time $O(T(n,m)+n\log U)$. Since the (IS) problem generalizes both the minimum cut problem and the finding of minima of n convex functions this time complexity is the best time complexity achievable. Any improvement in the run time of algorithms to identify the integer minima of convex functions or to find a minimum (parametric) cut would immediately translate into improvements of the run time of this algorithm.

10 Classes of quadratic problems solvable in strongly polynomial time

As noted earlier, the quadratic problem takes a special place among nonlinear optimization problems over linear constraints. This is because the optimality conditions are linear, and the solution to a system of linear inequalities is of polynomial length in the size of the coefficients. So for quadratic problems an optimal continuous solution is of length that is a polynomial function of the length of the input. In addition, the proof of impossibility for strongly polynomial algorithms using the algebraic tree computation model, is not applicable to the quadratic case. Still, recall that for the comparison computation model the proof is valid and it is impossible to derive strongly polynomial algorithms using only comparisons.

Only a 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 convex cost network flow problem in strongly polynomial time. The few results described here add to the limited repertoire of quadratic problems solved in strongly polynomial time.

10.1 Quadratic network flow problems

The feature that is common to all the techniques that have been used to derive strongly polynomial algorithms for quadratic separable problems is the use of a parametric search in order to solve the continuous problem. Then proximity is used to derive an integer solution. Several results pertaining to the problem of minimizing a *concave* problem are using parametric search to establish polynomiality and strong polynomiality when the network contains only a fixed number of nonlinear arc costs (Värbrand et al. 1995, 1996) or when the network has some special properties (e.g. production-transportation problem with a a transportation matrix with Monge property Hochbaum and Hong 1996). We review here some classes of quadratic convex network flow problems that can be solved in strongly polynomial time.

Simple quadratic allocation in linear time Brucker (1984) described a linear time algorithm for the quadratic continuous SRA. Our adaptation of the algorithm, and its application to the integer case is described next.

The algorithm for the quadratic resource allocation problem, (QRA), is based on a search for an optimal Lagrange multiplier. The continuous QRA is formulated as follows:

$$\begin{aligned} \text{(QRA)} \quad \min \quad & \sum_{i=1}^{n}\left[a_i x_i + \frac{1}{2}b_i x_i^2\right] \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = d, \\ & x_i \geq 0, \quad i = 1,\ldots,n, \end{aligned}$$

where d is positive (what earlier was denoted by B) and each b_i is positive.

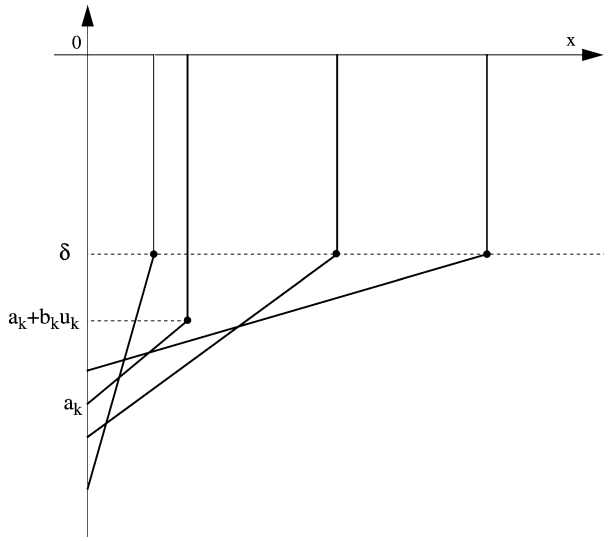

Fig. 2 The quadratic resource allocation QRA

The convexity of the objective function guarantees that a solution satisfying the Kuhn–Tucker conditions is also optimal. In particular, we seek a non-negative solution $\mathbf{x}^*$ and a value δ^* such that:

$$\sum_{i=1}^{n} x_i^* = d \quad \text{and,}$$

$$x_i^* > 0 \quad \text{implies that} \quad a_i + b_i x_i^* = \delta^*.$$

The situation is illustrated in Fig. 2. The value set for δ determines associated values for x_i. For any value δ, the associated solution $\mathbf{x}$ is:

$$x_i = 0 \quad \text{for } i \text{ such that } a_i > \delta,$$

$$x_i = \frac{\delta - a_i}{b_i} \quad \text{for } i \text{ such that } a_i \leq \delta.$$

Finding the optimal solution to QRA is equivalent to finding a value δ^* such that the associated solution satisfies $\hat{d} = \sum_{i=1}^{n} x_i$ is equal to d. If $\hat{d} < d$, then we could conclude that δ^* is greater than δ, because any smaller value would yield an even smaller value for $\hat{d}$. Similarly, if $\hat{d} > d$, then δ^* is less than δ.

For any δ, the value of $\hat{d}$ is dependent on the coefficients in the set $\{i \mid a_i > \delta\}$. Consequently, $\hat{d}(\delta)$ is a monotone, piecewise linear function having breakpoint values $a_i, i = 1, \ldots, n$. Its monotonicity allows for a binary search for the optimal value, δ^*, satisfying $\hat{d}(\delta) = d$.

Since the value of d is finite then there is a finite optimal δ^* for every instance of QRA. The algorithm we propose for finding δ^*, chooses "guesses" (from among the breakpoint values, a_i), until it finds two consecutive breakpoints which contain δ^* in the interval between them. In this range, $\hat{d} = \sum_i x_i$ is a linear function in δ. The problem is then solved by finding the particular value of δ for which $\hat{d} = d$, (i.e., by solving the linear equation in one variable).

In the algorithm the parameters A and B maintain partial sums necessary to evaluate $\sum_{i=1}^{n} x_i$, without computing the sum at every iteration from scratch.

Procedure QRA:

Step 0: Let $L = \{a_1, \ldots, a_n\}$, $I = \{1, \ldots, n\}$, $A = \sum_{i=1}^{n} \frac{a_i}{b_i}$, $B = \sum_{i=1}^{n} \frac{1}{b_i}$.

Step 1: Set $\delta \leftarrow a_m$, the median value from the set L. Let $\hat{A} = A - \sum_{i \in I^+} \frac{a_i}{b_i}$, and $\hat{B} = B - \sum_{i \in I^+} \frac{1}{b_i}$, where $I^+ = \{i \in I \mid a_i > \delta\}$.

Step 2: Let $\hat{d} = \hat{B}\delta - \hat{A}$.
If $\hat{d} = d$ then output "$\delta = \delta^*$" and stop.
If $\hat{d} > d$ then $\delta > \delta^*$.
If $\hat{d} < d$ then $\delta < \delta^*$.

Step 3: If $\delta > \delta^*$ then set $I \leftarrow \{i \in I \mid a_i < \delta\}$, $L \leftarrow \{a_i \mid i \in I\}$, $A \leftarrow \hat{A} - \frac{a_m}{b_m}$, $B \leftarrow \hat{B} - \frac{1}{b_m}$.
Else, $\delta < \delta^*$ and set $I \leftarrow \{i \in I \mid a_i \geq \delta\}$, $L \leftarrow \{a_i \mid i \in I\}$.

Step 4: If $|L| \geq 2$, go to Step 1.
Else $\delta^* = \frac{d+A}{B}$.

The algorithm outputs a value δ^*. The optimal solution $\mathbf{x}^*$ is then readily available, and can be determined in linear time:

$$x_i^* = \begin{cases} \delta^* - \frac{a_i}{b_i} & \text{for } i \text{ such that } a_i \leq \delta^*, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 6 (Cosares and Hochbaum 1994) *Procedure* QRA *finds* δ^* *and* $\mathbf{x}^*$ *in* $O(n)$ *time.*

Proof For any guess δ, the values of $\hat{A}$ and $\hat{B}$ are set to assure that $\hat{d} = \sum x_i$ is set to the appropriate value (i.e. $x_i = 0$ when $a_i > \delta$). The element a_i is removed from L if either it is known to be greater than δ^* or if it is less than an established lower bound for δ^*. When L contains only one element, say a_i, then we can conclude that δ^* is between a_i and a_j, the next largest of the a's. Furthermore, since $\hat{d}$ is a linear function of δ in this range, (i.e. $\hat{d} = \hat{B}\delta - \hat{A}$), δ^* and $\mathbf{x}^*$ are determined as in Step 4.

The $O(n)$ complexity of the algorithm follows from the fact that each of Steps 1, 2 and 3 can be performed in a number of arithmetic operations that is linear in the cardinality of the set L, including the selection of the median value Blum et al. (1972). Since the number of elements in the set is initially n and is cut in half after each pass, the total work is linear in $(n + n/2 + n/4 + \ldots) \leq 2n$, so the complexity of the algorithm is $O(n)$. □

When the problem is to be solved in integers we apply the proximity theorem for the general allocation problem, Theorem 2. From the optimal continuous solution $\mathbf{x}^*$ we create a lower bound vector to the optimal integer solution, $\mathbf{x}^* - \mathbf{e}$. Since $\sum x_j^* = d$, there are only n more units to add which can be determined in as many iterations as the greedy algorithm, each taking a constant time. The running time is therefore linear for the integer version of the problem.

A similar, though slightly more complex, algorithm works to solve QRA where each variable has an upper bound. A linear time procedure is described in Hochbaum and Hong (1995).

Quadratic allocation flow problem The *network* allocation problem is a special case of (GAP) as proved in Federgruen and Groenevelt (1986a, 1986b). As such it is solvable in

pseudo-polynomial time by the greedy algorithm, and by a polynomial time algorithm with the proximity-scaling procedure described in Sect. 5. The problem is defined on a network with a single source and a set of sinks.

All known special cases of the *network* allocation problem are solved very efficiently and in strongly polynomial time. These problems include the *network* allocation problem, the *tree* allocation problem, the *nested* allocation problem and the *generalized upper bounds* allocation problem. In Hochbaum and Hong (1995) we describe algorithms for these problems with respective complexities, $O(mn \log \frac{n^2}{m})$, $O(n \log n)$, $O(n \log n)$ and $O(n)$. These algorithms are all based on an efficient search for the optimal Lagrange multipliers and in that sense generalize the procedure *quadratic-allocation*.

We define the network allocation problem on a directed network $G = (V, A)$ with a single source node $s \in V$ and $T \subseteq V$ a set of sinks. Let $B > 0$ be the total supply of the source,and let C_{uv} be the capacity limit on each arc (u, v). Let the vector of the flow be $\phi = (\phi_{uv} : (u, v) \in A)$.

$$
\begin{aligned}
\text{(Net-alloc)} \quad \min \quad & \sum_{j \in T} \left(a_j x_j + \frac{1}{2} b_j x_j^2 \right) \\
\text{(i)} \quad & \sum_{(v,u) \in A} \phi_{vu} - \sum_{(u,v) \in A} \phi_{uv} = 0, \quad v \in V - T - \{s\}, \\
\text{(ii)} \quad & \sum_{(s,u) \in A} \phi_{su} - \sum_{(u,s) \in A} \phi_{us} \leq B, \\
\text{(iii)} \quad & \sum_{(u,j) \in A} \phi_{uj} - \sum_{(j,u) \in A} \phi_{ju} = x_j, \quad j \in T, \\
\text{(iv)} \quad & 0 \leq \phi_{uv} \leq C_{uv}, \quad (u, v) \in A, \\
& 0 \leq x_j \leq u_j, \quad \text{integer}, \ j \in T.
\end{aligned}
$$

The total sum of flow leaving the source B cannot exceed the minimum cut in the network. Also, as long as each variable is bounded in an interval where the derivative is negative and the sum of upper bounds is at least as large as B, the amount of flow in the network will be *equal* to B. So subject to such preprocessing the problem can be stated either with an equality or an inequality constraint on the source, (ii).

Net-alloc is not the same problem as a quadratic cost flow problem. In the latter problem there is an underlying network with a quadratic cost associated with the flow along *each* arc. In Net-alloc there is also an underlying network, but costs (quadratic) are associated only with the net flow at each sink. For this purpose we add a new dummy sink, t, and send all flows from the set of sinks T to that node. The costs are then only associated with arcs adjacent to node t. This graph is described in Fig. 3, where only the dashed lined arcs, that connect the sink to the 'variable' nodes, have costs associated with them.

Net-alloc is solvable, as described in Hochbaum and Hong (1995), in strongly polynomial time, $O(mn \log \frac{n^2}{m})$. The general idea of the algorithm is to establish the equivalence of the problem to a lexicographic flow problem. That latter problem is then posed as a parametric flow problem. That parametric flow problem has arc capacities which are each piecewise linear with a single breakpoint. We then generate all the breakpoints of the function using an algorithm that extends the algorithm by Gallo et al. (1989) for parametric flow problems. (Their algorithm is applicable when each arc capacity is *linear*.) As such, this algorithm, like the others in this section, is based on the concept of parametric search.

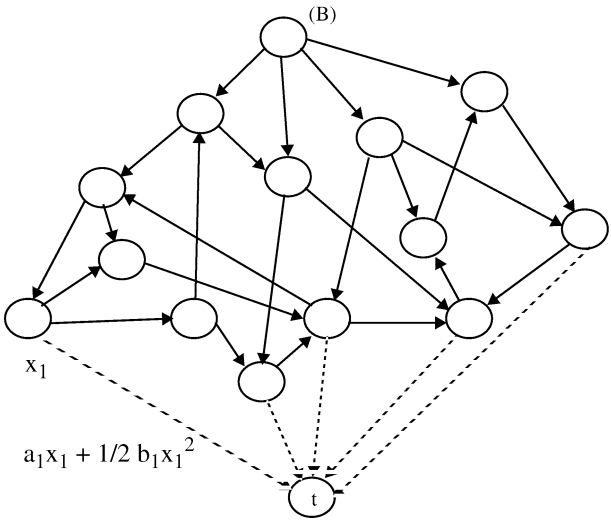

Fig. 3 The quadratic network allocation problem

Quadratic transportation with fixed number of suppliers Using a known transformation, any minimum cost network flow problem can be formulated as a transportation problem (see e.g. Ahuja et al. 1993). It is therefore sufficient in the search for efficient algorithms for the quadratic separable network flow problem to focus on the quadratic separable transportation problem.

The quadratic transportation problem (QTP) is defined on a bipartite network, with k supply nodes and n demand nodes. The cost of transporting flow from a supply node to a demand node is a convex quadratic function of the flow quantity. The formulation of the continuous problem is as follows:

$$
\begin{aligned}
\text{(QTP)} \quad \min \quad & \sum_{i=1}^{k}\sum_{j=1}^{n}\left[a_{ij}x_{ij} + \frac{1}{2}b_{ij}x_{ij}^2\right] \\
\text{s.t.} \quad & \sum_{j} x_{ij} = s_i, \quad i = 1,\ldots,k, \\
& \sum_{i} x_{ij} = d_j, \quad j = 1,\ldots,n, \\
& x_{ij} \geq 0, \quad i = 1,\ldots,k,\ j = 1,\ldots,n.
\end{aligned}
$$

where $b_{ij} > 0$, $s_i > 0$, and $d_j > 0$ are rational numbers and $\sum_i s_i = \sum_j d_j$.

While it is not known whether the QTP is solvable in strongly polynomial time, Cosares and Hochbaum (1994) gave a strongly polynomial algorithm for the case when the number of supply nodes k is fixed. That algorithm exploits the relationship between the transportation problem and the allocation problem. The continuous allocation problem can be solved by identifying a Lagrange multiplier associated with the single constraint. In the quadratic case this can be done in linear time. The algorithm for the QTP entails relaxing and aggregating supply constraints, and then searching for optimal values for the Lagrange multipliers. For the case of two supply nodes, $k = 2$, the algorithm is linear. For greater values of k, the algorithm has running time of $O(n^{k+1})$. A result by Megiddo and Tamir (1993), which

invokes an alternative searching method, yields a linear running time of the algorithm for fixed k (where the constant coefficient is exponential in k).

Quadratic separable flow on series-parallel graphs Tamir (1993) devised a strongly polynomial algorithm for minimum convex quadratic separable cost flow when the underlying network is *series-parallel*, in the presence of a single source-sink pair. The cost of the flow is viewed as a parametric function of the available supply. The algorithm is exploiting the series-parallel structure of the network in order to construct an optimal continuous solution. A series-parallel graph is constructed recursively from two smaller series-parallel graphs using two types of composition operations:

1. *series*, where one graph identifies its source with the sink of the other, or
2. *parallel*, where the two graphs identify their source nodes and sink nodes as one.

The value of the cost function for the series composition is the sum of the cost functions for each one of the graphs. For parallel composition, the combination is a solution to an optimization function for all possible partitions of the flow into the two parallel graphs. This optimization function is in fact a SRA and some of its properties are used to derive a solution in $O(m^2)$ time.

An integer solution is then determined from the continuous optimal solution using the same approach as described above. This is therefore another example where an optimal continuous solution is easier to determine than an integer one.

10.2 Quadratic knapsack problem

In the quadratic knapsack problem, the functions f_j are quadratic concave and g_j are linear. The optimal continuous solution in this case is of polynomial length in the size of the input. Thus there is an accuracy ϵ of polynomial length so that if a solution is optimal and ϵ-accurate, then the solution is also the exact optimal continuous solution.

The quadratic continuous knapsack problem is known to be solvable in linear time Brucker (1984). An alternative algorithm for solving the continuous quadratic knapsack problem is to reduce the problem to a QRA. For the specified accuracy ϵ we duplicate each entry $\frac{1}{\epsilon a_{ij}}$ times. ϵ is chosen so that any solution that is ϵ-accurate is also optimal. The resulting quadratic allocation problem is solved using the linear time algorithm in Cosares and Hochbaum (1994) (there is one supply node and therefore $k = 1$).

10.3 The quadratic CCC and s-excess problems

In the quadratic case our algorithm described in Sect. 9 is implemented to run in strongly polynomial time. This is easily achieved since the derivative functions are linear—a case shown in Gallo et al. (1989) to be solvable 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, $O(mn \log \frac{n^2}{m})$.

11 The complexity of some nonseparablecases

Strong polynomiality of continuous nonseparable problems Even though separable convex quadratic problems may be solvable in strongly polynomial time, the question of strong polynomiality of *nonseparable* quadratic continuous optimization problems is open. While

it is possible that the nonseparable optimization problem is solvable in strongly polynomial time, establishing that is at least as hard as the question of strong polynomiality of linear programming (this insight is due to I. Adler). To see this, observe that the problem of feasibility of linear programming $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\} \neq \emptyset$? is equivalent to the following quadratic nonseparable convex problem subject only to nonnegativity constraints:

$$\begin{aligned} \min \quad & (A\mathbf{x} - \mathbf{b})^T (A\mathbf{x} - \mathbf{b}) \\ & \mathbf{x} \geq 0. \end{aligned}$$

Since this question is not resolved, we do not investigate it here, as it should be treated in the framework of the strong polynomiality of linear programming.

The polynomiality of continuous convex nonseparable flow problems *Nonseparable* convex *continuous* problems, as well as nonseparable quadratic convex continuous problems are solvable in polynomial time as follows. A solution approximating the optimal *objective* value to the convex continuous problem is obtainable in polynomial time, provided that the gradient of the objective functions are available and that the value of the optimal solution is bounded in a certain interval. Such work, based on the Ellipsoid method, is described by Nemirovsky and Yudin (1983). In the quadratic case, exact solutions are possible. Indeed, the polynomial solvability of continuous convex quadratic programming problems over linear constraints was established as a byproduct of the ellipsoid algorithm for linear programming (see Kozlov et al. 1979). The best running time reported to date is by Monteiro and Adler Monteiro and Adler (1989), $O(m^3 L)$, where L represents the total length of the input coefficients and m the number of variables. Similar results were also given by Kapoor and Vaidya (1986). Note that these running times are not strongly polynomial.

The NP-completeness of integer quadratic nonseparable problems The case for the *integer* problems that are *nonseparable*, even if convex, is harder. Nonseparable quadratic *integer* problems are NP-hard, To see this consider the following known reduction from the independent set problem. The maximization of the weight of an independent set in a graph is formulated as follows: Given a graph $G = (V, E)$ with nonnegative weights W_v for each $v \in V$, find a subset of vertices $U \subseteq V$ such that for any $i, j \in U$, $\{i, j\} \notin E$, and such that the total weight $W(U) = \sum_{v \in U} W_v$ is maximum. The weighted independent set problem can be posed as the quadratic maximization problem:

$$\begin{aligned} \max \quad & \sum_{v \in V} W_v x_v - \sum_{\{u,v\} \in E} W(V) \cdot x_u \cdot x_v \\ & x_v \in \{0, 1\}. \end{aligned}$$

Let $\mathbf{x}^*$ be the optimal solution. The maximum weight independent set is then $\{v \mid x_v^* = 1\}$. Note that the reduction also applies for the unweighted case. So even in the absence of the flow balance constraints, the integer problem is NP-hard. The objective function in this case is not necessarily concave. The question is then asked whether the complexity of the problem is not a result of the indefiniteness of the quadratic matrix.

The answer is negative as we demonstrate now. Consider the quadratic minimization problem,

$$\begin{aligned} \min \quad & \mathbf{x}^T Q \mathbf{x} - \mathbf{d}^T \mathbf{x} \\ & \mathbf{x} \in \{0, 1\}^n. \end{aligned}$$

Baldick (1991) proved that this problem with Q having nonnegative off-diagonal elements is NP-hard. Any convex quadratic minimization is therefore also NP-hard. The proof is by reduction from the set splitting problem (see Garey and Johnson 1979) as follows. Let E be a collection of 2 and 3-element subsets $E_1, \ldots, E_N$ of $\{1, \ldots, n\}$ with $N \leq \frac{1}{6}(n+1)n(n-1)$. The set splitting problem is to find a set $Z \subset \{1, \ldots, n\}$ such that no E_i is contained in Z nor disjoint from it. In that case Z is said to satisfy the splitting property. Deciding whether such a Z exists is NP-complete, Garey and Johnson (1979).

Consider the following quadratic function,

$$f(\mathbf{x}) = \sum_{i=1}^{N} \left(\sum_{j \in E_i} (2x_j - 1) \right)^2 - \sum_{i=1}^{N} |E_i|^2.$$

$f(\mathbf{x})$ is of the form $\mathbf{x}^T Q\mathbf{x} - \mathbf{d}^T\mathbf{x}$ with Q positive definite with positive off-diagonal elements of magnitude bounded in N.

For k_3 the number of 3-element subsets in E, $f(\mathbf{x}) \leq k_3 - \sum_{i=1}^{N} |E_i|^2$ if and only if $Z = \{j \mid x_j = 1\}$ satisfies the splitting property. In particular, it is NP-hard to find the minimum of f. Hence the nonseparability is the factor that makes this problem hard.

Polynomial cases of nonseparable integer quadratic problems—the "separating scheme" We illustrate one general purpose technique for nonseparable problems that we call a "separating scheme". The technique relies on converting the objective function into a separable function (e.g. by diagonalizing the matrix Q in the quadratic case). This implies a transformation of variables which affects the constraints. If the new constraints are such that they form a totally unimodular matrix then the proximity-scaling algorithm by Hochbaum and Shanthikumar (1990) for separable convex optimization over totally unimodular constraints can be employed to obtain an optimal integer solution. This proximity-scaling algorithm solves, at each iteration, the scaled problem in integers using linear programming.

Consider the nonseparable problem:

$$\begin{aligned} \min \quad & F(\mathbf{x}) \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{u}, \\ & \mathbf{x} \in Z^n. \end{aligned}$$

Suppose there exists an invertible $n \times n$ matrix U such that $F(U\mathbf{x})$ is a separable function. Then the newly stated problem is:

$$\begin{aligned} \min \quad & F(\mathbf{y}) \\ \text{s.t.} \quad & AU^{-1}\mathbf{y} = \mathbf{b}, \\ & \mathbf{0} \leq U^{-1}\mathbf{y} \leq \mathbf{u}, \\ & U^{-1}\mathbf{y} \quad \text{integer}. \end{aligned}$$

Now, if the matrix U is totally unimodular, then the integrality requirement is preserved. If the new matrix of constraints $\begin{bmatrix} AU^{-1} \\ U^{-1} \end{bmatrix}$ is totally unimodular, then the nonseparable flow problem is solvable in polynomial time and in integers using the proximity-scaling algorithm.

Consider now the separating scheme for quadratic objective functions. The formulation of a quadratic nonseparable problem on box constraints is as follows:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} d_i x_i + \sum q_{ij} \cdot x_i \cdot x_j \\ & \ell_i \le x_i \le u_i, \\ & x_i \quad \text{integer}. \end{aligned}$$

The idea of making the problem separable so that the resulting constraint matrix is totally unimodular is translated here to finding a totally unimodular matrix U, so that for the matrix $Q = (q_{ij})$, $U^{-1}QU$ is a diagonal matrix. Baldick and Wu (1990) used this approach for a problem of electric distribution systems where only box constraints are present.

Baldick (1991) has further identified several classes of matrices Q where a "diagonalizing" scheme with a totally unimodular matrix exists. The two classes are:

(1) Diagonally dominant matrices, $q_{ii} \ge \sum_{i \ne j} |q_{ij}|$.
(2) Matrices with forest structure: These are matrices with a partial order on the positive coefficients inducing a forest.

For both these classes, with A empty, there are polynomial algorithms. Also if the constraint matrix, $\left[\begin{smallmatrix} AU^{-1} \\ U^{-1} \end{smallmatrix}\right]$ is totally unimodular then still integer solutions can be obtained in polynomial time. Continuous solutions can be obtained in polynomial time if the largest subdeterminant of the constraint matrix is bounded by a polynomial (Hochbaum and Shanthikumar 1990).

Miscellaneous polynomial cases Barahona (1986) proved that quadratic nonseparable 0-1 optimization is polynomially solvable if the quadratic matrix Q has a series-parallel graph characteristic structure. That is, there exists a series-parallel graph $G = (V, E)$ with $q_{ij} \ne 0$ if and only if $(i, j) \in E$. The algorithm involves transforming the problem into a maximum cut problem which is in turn solved recursively using the fact that the underlying graph is series-parallel.

A class of nonseparable problems over box constraints is solvable in strongly polynomial time if in the objective $\min \mathbf{x}^T Q\mathbf{x} - \mathbf{d}^T\mathbf{x}$ all elements of Q are nonpositive. This type of problem is solvable by transforming it into the selection problem and hence a minimum cut problem on a bipartite network. This transformation however is not a separating scheme as it is a nonlinear transformation. Hochbaum (1989) has generalized this class to include all "bipartite polynomials". This generalized class is identified by the property of the multivariate polynomial objective function called the *bipartition property*. This property can be easily described on a graph for quadratic objective functions: $G = (V, E)$ with $V = V_1 \cup V_2$ so that $q_{ij} > 0$ only if both $i, j \in V_1$ or $i, j \in V_2$. This property was discovered independently, for quadratic objective functions, by Hansen and Simeone (1986). In Hansen and Simeone (1986), an objective function with this property is called a *unate function*. With this property a modified reduction still works to transform the problem into a minimum cut problem which is then solved in polynomial time.

In Hochbaum et al. (1992) a "high multiplicity" minimum weighted tardiness scheduling problem was discussed. This problem was formulated as a quadratic transportation problem with a nonseparable objective function. This problem is unique among the problems discussed in this section in that the set of constraints is not empty. In that problem, the right hand sides (supplies and demands), and the linear coefficients in the objective function are

large, so the aim was to find an algorithm for the integer case the running time of which is independent of these numbers. Such algorithm was found by solving a related continuous problem (not a relaxation), the solution of which could be rounded using a simple procedure to derive an optimal integer solution.

All problems presented in this section are special classes. There is still a need to discover the extent of the polynomial solvability of nonseparable network flow problems, although this cannot be expected to be so unified as in the separable case.

12 Conclusions and open problems

We survey in this paper a collection of results pertaining to nonlinear optimization problems. Several classes of nonlinear problems, such as concave separable or convex nonseparable problems, are NP-hard and the emphasis is on developing algorithms for polynomial subclasses. For the convex separable flow problem there are polynomial algorithms and even lower bounds indicating the impossibility of strongly polynomial algorithms for the nonquadratic instances.

This work leaves a number of questions unanswered. The major ones among these are:

1. For convex quadratic separable problems either prove a lower bound that proves the impossibility of strongly polynomial algorithms or identify a strongly polynomial algorithm. We conjecture that the latter is possible.
2. For the well solvable case of convex separable network flow, improve the capacity scaling algorithm to an algorithm that depends on the double logarithm of B rather than on the logarithm of B. This may involve techniques borrowed from those used to find roots for a system of polynomials.
3. Delineate and generalize the largest possible subclasses of nonseparable convex cases that are solvable in polynomial time. In particular there has been little research involving such problems with a nonempty set of flow balance constraints. As a result there is little insight to the behavior of network flow optimal solutions in the presence of such nonseparable costs.
4. Tighten proximity theorems or find threshold theorems for classes of problems other than the ones reported here, with the resulting improved algorithms for solving the problems.

Acknowledgement This research has been supported in part by NSF award No. DMI-0620677.

References

Ahuja, R. K., & Orlin, J. B. (2001b). Inverse optimization. *Operations Research*, *49*, 771–783.

Ahuja, R. K., Batra, J. L., & Gupta, S. K. (1984). A parametric algorithm for the convex cost network flow and related problems. *European Journal of Operational Research*, *16*, 222–235.

Ahuja, R. K., Hochbaum, D. S., & Orlin, J. B. (2003). Solving the convex cost integer dual network flow problem. *Management Science*, *49*, 950–964.

Ahuja, R. K., Hochbaum, D. S., & Orlin, J. B. (2004). A cut based algorithm for the nonlinear dual of the minimum cost network flow problem. *Algorithmica*, *39*, 189–208.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms and applications*. New Jersey: Prentice Hall.

Baldick, R. (1991). *A unification of polynomially solvable cases of integer 'non-separable' quadratic optimization*. Lawrence Berkeley Laboratory manuscript.

Baldick, R., & Wu, F. F. (1990). Efficient integer optimization algorithms for optimal coordination of capacitors and regulators. *IEEE Transactions on Power Systems*, *5*, 805–812.

Barahona, F. (1986). A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics*, *13*, 23–28.

Barlow, R. E., Bartholomew, D. J., Bremer, J. M., & Brunk, H. D. (1972). *Statistical inference under order restrictions*. New York: Wiley.

Blum, M., Floyd, R. W., Pratt, V. R., Rivest, R. L., & Tarjan, R. E. (1972). Time bounds for selection. *Journal of Computer Systems Science*, *7*, 448–461.

Brucker, P. (1984). An *O(n)* algorithm for quadratic knapsack problems. *Operations Research Letters*, *3*, 163–166.

Burton, D., & Toint, Ph. L. (1992). On an instance of the inverse shortest paths problem. *Mathematical Programming*, *53*, 45–61.

Burton, D., & Toint, Ph. L. (1994). On the use of an inverse shortest paths algorithm for recovering linearly correlated costs. *Mathematical Programming*, *63*, 1–22.

Busacker, R. G., & Gowen, P. J. (1961) *A procedure for determining minimal-cost network flow patterns*. Operational Research Office, John Hopkins University, Baltimore, MD.

Cosares, S., & Hochbaum, D. S. (1994). A strongly polynomial algorithm for the quadratic transportation problem with fixed number of suppliers. *Mathematics of Operations Research*, *19*, 94–111.

Dantzig, G. B. (1963). *Linear programming and extensions*. New Jersey: Princeton University Press.

Dennis, J. B. (1959). Mathematical programming and electrical networks. In *Technology press research monographs* (pp. 74–75). New York: Technology Press and Wiley.

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, *19*, 248–264.

Erickson, R. E., Monma, C. L., & Veinott, A. F. (1987). Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, *12*, 634–664.

Federgruen, A., & Groenevelt, H. (1986a). The greedy procedure for resource allocation problems: Necessary and sufficient conditions for optimality. *Operations Research*, *34*, 909–918.

Federgruen, A., & Groenevelt, H. (1986b). Optimal flows in networks with multiple sources and sinks, with applications to oil and gas lease investment programs. *Operations Research*, *34*, 218–225.

Frederickson, G. N., & Johnson, D. B. (1982). The complexity of selection and rankings in $X + Y$ and matrices with sorted columns. *Journal of Computing System Science*, *24*, 197–208.

Fourer, R. (1988). A simplex algorithm for piecewise-linear programming: Finiteness, feasibility and degeneracy. *Mathematical Programming*, *41*, 281–316.

Gale, D. (1957). A theorem of flows in networks. *Pacific Journal of Mathematics*, *7*, 1073–1082.

Gallo, G., Grigoriadis, M. D., & Tarjan, R. E. (1989). A fast parametric maximum flow algorithm and applications. *SIAM Journal of Computing*, *18*, 30–55.

Garey, M., & Johnson, D. (1979). *Computers and intractability, a guide to the theory of NP-completeness*. New York: Freeman.

Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum flow problem. *Journal of the ACM*, *35*, 921–940.

Granot, F., & Skorin-Kapov, J. (1990). Some proximity and sensitivity results in quadratic integer programming. *Mathematical Programming*, *47*, 259–268.

Guisewite, G., & Pardalos, P. M. (1990). Minimum concave cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research*, *25*, 75–100.

Hansen, P., & Simeone, B. (1986). Unimodular functions. *Discrete Applied Mathematics*, *14*, 269–281.

Hochbaum, D. S. (1989). *On a polynomial class of nonlinear optimization problems*. Manuscript, U.C. Berkeley.

Hochbaum, D. S. (1993). Polynomial algorithms for convex network optimization. In D. Du, M. Pardalos (Eds.), *Network optimization problems: algorithms, complexity and applications* (pp. 63–92). Singapore: World Scientific.

Hochbaum, D. S. (1994). Lower and upper bounds for allocation problems. *Mathematics of Operations Research*, *19*, 390–409.

Hochbaum, D. S. (1995). A nonlinear knapsack problem. *Operations Research Letters*, *17*, 103–110.

Hochbaum, D. S. (1998). *The pseudoflow algorithm for the maximum flow problem*. Manuscript, UC Berkeley, revised 2003. Extended abstract in: Boyd & Rios-Mercado (Eds.), Lecture notes in computer science: Vol. 1412. The pseudoflow algorithm and the pseudoflow-based simplex for the maximum flow problem. Proceedings of IPCO98 (pp. 325–337), Bixby, June 1998. New York: Springer.

Hochbaum, D. S. (2001). An efficient algorithm for image segmentation, Markov random fields and related problems. *Journal of the ACM*, *48*, 686–701.

Hochbaum, D. S. (2002). *The inverse shortest paths problem*. Manuscript, UC Berkeley.

Hochbaum, D. S. (2003). Efficient algorithms for the inverse spanning tree problem. *Operations Research*, *51*, 785–797.

Hochbaum, D. S. (2005). Complexity and algorithms for convex network optimization and other nonlinear problems. *4OR*, *3*, 171–216.
Hochbaum, D. S., & Hong, S. P. (1995). About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Mathematical Programming*, *69*, 269–309.
Hochbaum, D. S., & Hong, S. P. (1996). On the complexity of the production-transportation problem. *SIAM Journal on Optimization*, *6*, 250–264.
Hochbaum, D. S., & Queyranne, M. (2003). The convex cost closure problem. *SIAM Journal on Discrete Mathematics*, *16*, 192–207.
Hochbaum, D. S., & Seshadri, S. (1993). The empirical performance of a polynomial algorithm for constrained nonlinear optimization. *Annals of Operations Research*, *43*, 229–248.
Hochbaum, D. S., & Shanthikumar, J. G. (1990). Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, *37*, 843–862.
Hochbaum, D. S., Shamir, R., & Shanthikumar, J. G. (1992). A polynomial algorithm for an integer quadratic nonseparable transportation problem. *Mathematical Programming*, *55*, 359–372.
Hoffman, A. J. (1960). Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In R. Bellman, M. Hall Jr. (Eds.), *Proceedings of Symposia in Applied Mathematics: Vol. X. Combinatorial analysis* (pp. 113–127). Providence: American mathematical Society.
Ibaraki, T., & Katoh, N. (1988). *Resource allocation problems: Algorithmic approaches*. Boston: MIT.
Iri, M. (1960). A new method of solving transportation network problems. *Journal of the Operations Research Society of Japan*, *3*, 27–87.
Jewell, W. S. (1958). *Optimal flow through networks*. Technical report No. 8, Operations research Center, MIT, Cambridge.
Kapoor, S., & Vaidya, P. M. (1986). Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the 18th symposium on theory of computing* (pp. 147–159).
Karzanov, A. V., & McCormick, S. T. (1997). Polynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM Journal on Computing*, *26*, 1245–1275.
Knuth, D. (1973). *The art of computer programming: Vol. 3. Sorting and searching*. Reading: Addison Wesley.
Kozlov, M. K., Tarasov, S. P., & Khachian, L. G. (1979). Polynomial solvability of convex quadratic programming. *Doklady Akad. Nauk SSSR*, *5*, 1051–1053 (Translated in *Soviet Mathematics Doklady* 20 (1979), 1108–1111).
Lawler, E. (1979). Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, *4*, 339–356.
Mansour, Y., Schieber, B., & Tiwari, P. (1991). Lower bounds for computations with the floor operation. *SIAM Journal on Computing*, *20*, 315–327.
Megiddo, N., & Tamir, A. (1993). Linear time algorithms for some separable quadratic programming problems. *Operations Research Letters*, *13*, 203–211.
Minoux, M. (1984). A polynomial algorithm for minimum quadratic cost flow problems. *European Journal of Operational Research*, *18*, 377–387.
Minoux, M. (1986). Solving integer minimum cost flows with separable convex cost objective polynomially. *Mathematical Programming Study*, *26*, 237–239.
Minoux, M. (1986). *Mathematical programming, theory and algorithms*. Wiley: New York, Chaps. 5, 6.
Monteiro, R. D. C., & Adler (1989). Interior path following primal-dual algorithms. Part II: Convex quadratic programming. *Mathematical Programming*, *44*, 43–66.
Moriguchi, S., & Shioura, A. (2004). On Hochbaum's proximity-scaling algorithm for the general resource allocation problem. *Mathematics of Operations Research*, *29*, 394–397.
Nemirovsky, A. S., & Yudin, D. B. (1983). *Problem complexity and method efficiency in optimization*. New York: Wiley.
Papadimitiou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: algorithms and complexity*. New Jersey: Prentice Hall.
Picard, J. C. (1976). Maximal closure of a graph and applications to combinatorial problems. *Management Science*, *22*, 1268–1272.
Pinto, Y., & Shamir, R. (1994). Efficient algorithms for minimum-cost flow problems with piecewise-linear convex costs. *Algorithmica*, *11*(3), 256–276.
Radzik, T. (1993). Parametric flows, Weighted means of cuts, and fractional combinatorial optimization. In P.M. Pardalos (Ed.), *Complexity in numerical optimization* (pp. 351–386). Singapore: World Scientific.
Renegar, J. (1987). On the worst case arithmetic complexity of approximation zeroes of polynomials. *Journal of Complexity*, *3*, 90–113.
Rote, G., & Zachariasen, M. (2007, to appear). Matrix scaling by network flow. In Proceedings of SODA07.
Sahni, S. (1974). Computationally related Problems. *SIAM Journal on Computing*, *3*, 262–279.

Shub, M., & Smale, S. (1996). Computational complexity: On the geometry of polynomials and a theory of cost, II. *SIAM Journal on Computing*, *15*, 145–161.

Sun, J., Tsai, K. -H., & Qi, L. (1993). A simplex method for network programs with convex separable piecewise linear costs and its application to stochastic transshipment problems. In D. Du & P. M. Pardalos (Eds.), *Network optimization problems: Algorithms, complexity and applications* (pp. 283–300). Singapore: World Scientific.

Tamir, A. (1993). A strongly polynomial algorithm for minimum convex separable quadratic cost flow problems on series-parallel networks. *Mathematical Programming*, *59*, 117–132.

Tardos, E. (1985). A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, *5*, 247–255.

Tardos, E. (1986). A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, *34*, 250–256.

Värbrand, P., Tuy, H., Ghannadan, S., & Migdalas, A. (1995). The minimum concave cost network flow problems with fixed number of sources and non-linear arc costs. *Journal of Global Optimisation*, *6*, 135–151.

Värbrand, P., Tuy, H., Ghannadan, S., & Migdalas, A. (1996). A strongly polynomial algorithm for a concave production-transportation problem with a fixed number of non-linear variables. *Mathematical Programming*, *72*, 229–258.