Available online at www.sciencedirect.com

ScienceDirect

Discrete Optimization 5 (2008) 550–561

DISCRETE OPTIMIZATION

www.elsevier.com/locate/disopt

ε-optimization schemes and L-bit precision: Alternative perspectives for solving combinatorial optimization problems[☆]

James B. Orlin[a], Andreas S. Schulz[a,*], Sudipta Sengupta[b]

[a] *Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, USA*
[b] *Microsoft Research, Redmond, WA, USA*

Received 28 November 2005; accepted 27 August 2007
Available online 22 October 2007

Dedicated to the memories of George B. Dantzig and Peter L. Hammer

Abstract

Motivated by the need to deal with imprecise data in real-world optimization problems, we introduce two new models for algorithm design and analysis. The first model, called the L-bit precision model, leads to an alternate concept of polynomial-time solvability. Expressing numbers in L-bit precision reflects the format in which large numbers are stored in computers today. The second concept, called ε-optimization, provides an alternative approach to approximation schemes for measuring distance from optimality. In contrast to the worst-case relative error, the notion of an ε-optimal solution is invariant under subtraction of a constant from the objective function, and it is properly defined even if the objective function takes on negative values.

Besides discussing the relation between these two models and preexisting concepts, we focus on designing polynomial-time algorithms for solving NP-hard problems in which some or all data are expressed with L-bit precision, and on designing fully polynomial-time ε-optimization schemes for NP-hard problems, including some that do not possess fully polynomial-time approximation schemes (unless P = NP).

MSC: primary 68Q17; 68Q25; 68W25; 68W40; 90C27; 90C60

Keywords: Approximation schemes; Combinatorial optimization; Computational complexity; Inverse optimization

1. Introduction

Our work has its origin in inverse optimization. Let $X \subseteq \mathbb{Z}^n$ denote the set of feasible solutions to a minimization problem. Given a solution $x^* \in X$ and an a priori estimated cost vector $c \in \mathbb{R}^n$, the inverse optimization problem is to identify another cost vector $d \in \mathbb{R}^n$ such that $dx^* \leq dx$ for all $x \in X$, and such that the deviation of d from c is minimal. One rationale for inverse optimization is that the parameters of real-world problems, such as cost or

☆ A preliminary version of this paper appeared in the Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, May 21–23, 2000, pp. 565–572.
* Corresponding author.
E-mail addresses: jorlin@mit.edu (J.B. Orlin), schulz@mit.edu (A.S. Schulz), sudipta@microsoft.com (S. Sengupta).

1572-5286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2007.08.004

capacity, are already approximate. Hence, one might consider x^* to be a sufficiently good solution if it is optimal with respect to a modestly perturbed cost vector. We refer to [2] for an overview of inverse optimization applied to linear and combinatorial optimization problems.

In inverse optimization, x^* is assumed to be given. In this paper, we determine optimal feasible solutions, but sometimes we also choose the cost coefficients at the same time. We develop inverse optimization into an alternative approach for producing near-optimal solutions. In the course of this, we propose an alternative to the worst-case relative error, which is commonly used in the design of approximation algorithms to measure the distance from optimality. We also arrive at a different notion of solving a problem in polynomial time, assuming that only a certain number of bits is known with certainty.

One of the most active areas in the theory of computing is the study of approximation algorithms. An approximation algorithm for a combinatorial optimization problem is a polynomial-time algorithm that outputs feasible solutions that are guaranteed to be close to optimal (see, e.g., [3,13,22]). Therein, the distance from optimality is typically measured by the worst-case relative error. A feasible solution x^* to an optimization problem $\min\{cx : x \in X\}$ has a worst-case relative error of at most ε if $(cx^* - cx)/cx \leq \varepsilon$ for all $x \in X$. This way of measuring the quality of an approximate solution faces some well-known, inherent drawbacks. In particular, if the optimal objective value is non-positive, the relative error is an inappropriate measure of performance. Moreover, a translation of variables has a dramatic impact on the relative error. That is, replacing x by $x - a$ will, in general, lead to very different measures of performance. We propose an alternative that overcomes these difficulties. A vector c' is an *ε-perturbation* of the vector c if the following is true, for some $0 < \varepsilon < 1$:

$$\begin{aligned} &c_j(1-\varepsilon) \leq c'_j \leq c_j(1+\varepsilon) && \text{if } c_j \geq 0, \quad \text{and} \\ &-c_j(1-\varepsilon) \leq -c'_j \leq -c_j(1+\varepsilon) && \text{if } c_j < 0. \end{aligned}$$

Note that $c'_j = 0$ if $c_j = 0$. Consider the optimization problem $\min\{cx : x \in X\}$.[1] We say that a solution x' is *ε-optimal* if there exists an ε-perturbation c' of c such that x' is optimal for the problem $\min\{c'x : x \in X\}$. An *ε-optimization algorithm* is an algorithm that produces an ε-optimal solution for every instance of that problem. Moreover, we say that a family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ is a *polynomial-time ε-optimization scheme* (PTEOS) for an optimization problem if for every fixed $\varepsilon > 0$, and for every instance (c, X) of the problem, the algorithm A_ε produces an ε-optimal solution in time that is polynomial in the size of the instance. We say that $(A_\varepsilon)_{\varepsilon>0}$ is a *fully polynomial-time ε-optimization scheme* (FPTEOS) if given a perturbation ε and given an instance, A_ε produces an ε-optimal solution in time polynomial in the size of the instance and in $1/\varepsilon$. In other words, the running time behavior of a PTEOS and an FPTEOS is precisely the same as that of a polynomial-time approximation scheme (PTAS) and a fully polynomial-time approximation scheme (FPTAS), respectively. However, the manner in which we measure proximity to the optimum is different.

We now introduce the *L-bit precision model*. In this model, some or all input numbers of the problem are of the form $a \cdot 2^t$, where a and t are integers, $t \geq 0$, and $|a| < 2^L$. We say that these numbers are expressed with L-bit precision. The space to store a number expressed with L-bit precision is assumed to be $O(L + \log t)$. If T is a tight upper bound on the exponents t of the integers expressed in L-bit precision, an algorithm is said to run in polynomial time if its running time is polynomial in n, the number of data items, and $\log T$; it is strongly polynomial-time if its running time is polynomial in n, but does not depend on T. Here, we assume either that L is constant or $L = O(\log n)$. In the former case, we speak about the *fixed precision model*, whereas the latter case is *logarithmic precision*.

Whenever we refer to a problem as being NP-hard or polynomial-time solvable without explicitly specifying the way in which numbers are represented, we assume that the input data are encoded in the standard way, i.e., by using their binary representation. In particular, the input size of a non-zero integer of the form $a \cdot 2^t$ under standard encoding is $\Theta(\log(a \cdot 2^t)) = \Theta(\log a + t)$. Notice that algorithms that run in polynomial time with respect to the standard encoding of numbers are not necessarily polynomial for the L-bit precision problem (because they may depend on T instead of being polynomial in $\log T$). On the other hand, if a problem is NP-hard in the strong sense, then it is NP-hard even if all data are of the order $O(n^k)$, for some fixed k. Consequently, if there were a polynomial-time algorithm under logarithmic precision, then P = NP.

[1] Although we restrict the following discourse to minimization problems, virtually all results extend in a natural way to the case of maximization problems.

Expressing numbers in L-bit precision is realistic because it incorporates the format in which large numbers are stored in computers today. In fact, one widely used format in the computer industry is the *IEEE Standard for Binary Floating-Point Arithmetic* [5,11,16]. Another reason why expressing numbers with L-bit precision seems realistic is that data are often known to several significant digits only in many practical application scenarios.

Main results

Our results fall into four categories. First, we design algorithms for NP-hard problems in which some or all data are expressed with L-bit precision. These algorithms run in polynomial time under the logarithmic or fixed precision regime. Second, we exhibit a connection between the L-bit precision model and ε-optimization algorithms. Third, we discuss the relation between fully polynomial-time ε-optimization schemes and fully polynomial-time approximation schemes, and we also provide additional general insights on ε-optimality. Finally, we design (fully) polynomial-time ε-optimization schemes for various combinatorial optimization problems, including some that do not have an (F)PTAS. The following overview details some of our findings:

1. For computation in the L-bit precision model, we present the following results:

(a) There is a strongly polynomial-time algorithm for solving the NP-complete knapsack problem under either the fixed or the logarithmic precision model; see Theorem 2.6.
(b) A simple variant of the knapsack problem is NP-complete even if the costs are expressed with fixed precision (Theorem 2.7).
(c) The NP-complete scheduling problem of minimizing the makespan of independent jobs on m identical parallel machines possesses a strongly polynomial-time algorithm under the L-bit precision model when m is fixed and L is fixed or logarithmic (Theorem 2.12).
(d) It follows from the strong NP-completeness of the 3-partition problem that it remains NP-complete under logarithmic precision. Nevertheless, there is a strongly polynomial-time algorithm for the 3-partition problem under the fixed precision model; see Theorem 2.15.

2. ε-optimization schemes can be linked to the L-bit precision model as follows. Suppose that the optimization problem $\min\{cx : x \in X\}$ is handed to us in L-bit precision and we are permitted to specify the next bit of any cost coefficient. In other words, we first replace every $c_j = a_j \cdot 2^{t_j}$ by $2a_j \cdot 2^{t_j}$. We may then replace $2a_j$ by $2a_j + 1$ if we want, or we may keep the coefficient as $2a_j \cdot 2^{t_j}$. The *bit rounding problem* consists of simultaneously choosing in the indicated way the $(L + 1)$st bit of precision for every cost coefficient, and finding an optimal solution for the resulting cost function. We show that, for many combinatorial optimization problems, a given ε-optimization scheme can be used to produce solutions to the bit rounding problem; see Theorem 3.1.

3. Whenever a combinatorial optimization problem, i.e., a problem $\min\{cx : x \in X\}$ with $X \subseteq \{0, 1\}^n$, has an FPTAS and is closed under fixing some of the variables to zero or one, then this problem also has an FPTEOS (Theorem 4.2). A problem that has this property is the knapsack problem, for instance. On the other hand, the existence of an FPTEOS also implies the existence of an FPTAS, provided that costs are non-negative (see Lemma 4.1).

4. A technique similar to the one used to turn an FPTAS into an FPTEOS also helps to design an FPTEOS for scheduling with rejection so as to minimize lateness (Theorem 5.1). This problem does not have an FPTAS. In fact, its objective function value can be non-positive.

Notice that ε-optimal solutions in its pure sense (as defined above) refer to perturbations in the objective function only. In order to accommodate problems with a different structure, like bin packing or some scheduling problems, we can extend the notion of ε-optimality by allowing for other input parameters of the problem to be changed as well. This gives rise to the following results. First, the strongly NP-hard bin packing problem has a PTEOS when we accept that item sizes may be perturbed; second, the strongly NP-hard scheduling problem of minimizing the makespan on identical parallel machines also has a PTEOS when job processing times may be slightly altered; see Theorem 5.2 for both results.

2. L-bit precision

In this section, we demonstrate that looking at problems under fixed or logarithmic precision can dramatically change their computational complexity. We consider three representative NP-hard combinatorial optimization problems – the knapsack problem, a scheduling problem, and the 3-partition problem – and design polynomial-time algorithms for them under L-bit precision.

2.1. The knapsack problem

An instance of the 0/1-knapsack problem consists of two sets of non-negative integers $w_1, w_2, \dots, w_n$ (the sizes of the items to be packed) and $c_1, c_2, \dots, c_n$ (their values), and a capacity b. The task is to determine an assignment of the binary variables $x_1, x_2, \dots, x_n$ to maximize $\sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} w_j x_j \le b$.

The knapsack problem is NP-hard [20], but it can be solved in pseudopolynomial time using either a dynamic programming recursion [4,6] or a closely related technique of reducing it to a shortest path problem in an expanded network $G = (N, A)$. We adopt the shortest path approach here. Let $C := \max_{j=1,2,\dots,n} c_j$. There is a source node, which we denote as $\langle 0, 0\rangle$. Moreover, for each $k \in \{1, 2, \dots, n\}$, and for each $v \in \{0, 1, \dots, nC\}$, there is a node $\langle k, v\rangle$, denoting a knapsack with items from the set $\{1, 2, \dots, k\}$ only and whose value is v. The network contains two kinds of arcs: there is an arc from node $\langle 0, 0\rangle$ to node $\langle 1, c_1\rangle$ of length w_1, and there is an arc from node $\langle 0, 0\rangle$ to node $\langle 1, 0\rangle$ with length 0. In general, for each $k \in \{2, 3, \dots, n\}$ and for each $v \in \{0, 1, \dots, nC\}$, there is an arc from $\langle k-1, v\rangle$ to $\langle k, v\rangle$ with length 0, and an arc from $\langle k-1, v\rangle$ to $\langle k, v + c_k\rangle$ of length w_k. The latter arc does not exist if $v + c_k > nC$. Let $s_k := \max\{v : \text{there is a path from node } \langle 0, 0\rangle \text{ to node } \langle k, v\rangle \text{ of length at most } b\}$. The following observations are straightforward, but useful.

Observation 2.1. *There is a path from $\langle 0, 0\rangle$ to $\langle k, v\rangle$ of length b' if and only if there are binary values $x_1, x_2, \dots, x_k$ such that $\sum_{j=1}^{k} w_j x_j = b'$ and $\sum_{j=1}^{k} c_j x_j = v$.*

Observation 2.2. *The optimal value for the knapsack problem is s_n.*

The above network is acyclic, and the shortest paths can be computed in $O(n^2 C)$ time (see, e.g., [1]), which is not polynomial in the size of the input under the L-bit precision model.

We now show how the computation associated with the above shortest path algorithm can be reduced to polynomial-time when the costs are represented with logarithmic or fixed precision. Assume that each c_j has an L-bit representation of the form $c_j = a_j \cdot 2^{t_j}$, with $a_j < 2^L$ for all $1 \le j \le n$. We first arrange the indices in non-increasing order of t_j's such that $t_1 \ge t_2 \ge \cdots \ge t_n$. Let C_k be the sum of the values of the last $n - k$ items; that is, $C_k := \sum_{j=k+1}^{n} c_j$. Let G^* denote the induced subgraph of G with node set $\{\langle k, v\rangle : 1 \le k \le n \text{ and } s_k - C_k < v \le s_k\} \cup \{\langle 0, 0\rangle\}$. Since no feasible packing of the last $n - k$ items can have a value larger than C_k, we get the following result.

Observation 2.3. *The network G can be replaced by G^* while maintaining an optimal solution for the knapsack problem.*

It turns out that the shortest path algorithm, if applied to G^*, runs in polynomial time when the cost coefficients are given in fixed or logarithmic precision. To make this precise, we need two additional observations.

Observation 2.4. *If there is a path from node $\langle 0, 0\rangle$ to node $\langle k, v\rangle$ in G^*, then v is an integer multiple of 2^{t_k}.*

Proof. This is an immediate consequence of the ordering of the indices and Observation 2.1. □

Corollary 2.5. *The number of nodes in G^* that are reachable from node $\langle 0, 0\rangle$ is at most $n^2 2^L$.*

Proof. The number of nodes $\langle k, v\rangle \in G^*$ for fixed value of k is at most C_k, which is bounded from above by $n 2^L 2^{t_{k+1}}$. Hence, the number of nodes for which v is an integer multiple of 2^{t_k} is at most $n 2^L$. The result follows. □

One concern with running the algorithm on G^* is that G^* is defined in terms of the values s_k for $k \in \{1, 2, \dots, n\}$, but the values s_k are not known a priori. In fact, they are determined in the course of the shortest path algorithm. This difficulty is overcome as follows: first, we observe that $s_k \le s_{k+1} \le s_k + c_{k+1}$. So, once the shortest path algorithm has scanned nodes at level k, it can determine s_{k+1} in at most 2^L steps, since there are at most 2^L values of v such that $v \in [s_k, s_k + c_{k+1}]$ and v is an integer multiple of $2^{t_{k+1}}$. Once s_{k+1} is determined, one can determine all nodes at level $k + 1$ in G^* that are potentially reachable from node $\langle 0, 0\rangle$ in $O(n 2^L)$ additional steps. This yields the following result.

Theorem 2.6. *The knapsack problem can be solved in* $O(n^2 2^L)$ *time when the objective function coefficients* c_j *are given in* L*-bit precision. This is strongly polynomial-time when* L *is fixed or logarithmic, i.e., under the fixed and logarithmic precision models.*

In contrast, we now present a modest generalization of the knapsack problem, which we call the group knapsack problem, that remains NP-complete even if profits and weights are expressed with 1-bit precision. In the decision formulation of the group knapsack problem, we are given non-negative integers $w_1, w_2, \dots, w_n$, $c_1, c_2, \dots, c_n$, c^*, and b, and a set A of pairs of indices. The question is whether there is an assignment of the binary variables $x_1, x_2, \dots, x_n$ such that $\sum_{j=1}^n w_j x_j \le b$, $\sum_{j=1}^n c_j x_j \ge c^*$, and $x_i = x_k$ for all $(i, k) \in A$.

The group knapsack problem is the same as the knapsack problem except that for each pair $(i, k) \in A$, either elements i and k are both placed in the knapsack or neither element is placed in the knapsack. The group knapsack problem is easily transformed into the knapsack problem as follows. Construct an undirected graph $G = (N, A)$ with node set $N = \{1, 2, \dots, n\}$ and an (undirected) arc set A (this is the same as the set A given in the problem). For each connected component of G with node set S, create a single item with value $\sum_{j \in S} c_j$ and weight $\sum_{j \in S} w_j$. These items form the input for the knapsack problem. Yet, the group knapsack problem remains NP-hard even if profits and weights are specified with just 1-bit precision. Note that 1-bit precision means that every objective function coefficient is a power of two.

Theorem 2.7. *The* 1*-bit version of the group knapsack problem is NP-complete.*

Proof. The n-bit version of the knapsack problem is known to be NP-complete. (We actually assume that the items have values between 0 and $2^n - 1$.) We can transform any n-bit version of the knapsack problem into a 1-bit version of the group knapsack problem by replacing each element j of the original knapsack problem by a group Group(j) of at most n elements such that the sum of the values of Group(j) is c_j, the sum of the weights of Group(j) is w_j, and each integer weight and value is a power of 2. The values of the elements of Group(j) are derived from the binary representation of c_j, and the weights of the elements of Group(j) are derived from the binary representation of w_j. □

Theorems 2.6 and 2.7 illustrate that two problems of identical behavior under the standard representation of numbers can display a very different computational complexity in terms of L-bit precision encoding.

2.2. Parallel machine scheduling

In this section, we consider the problem of scheduling n jobs on m identical parallel machines. We address the following recognition variant of the makespan problem:

> Given a set of n independent jobs with integer processing times $p_1, p_2, \dots, p_n$ and m parallel machines with capacity b_i on machine i, is there a way of assigning jobs to machines such that the total processing time on machine i is at most b_i, for all $i = 1, 2, \dots, m$?

This problem is already NP-hard for $m = 2$ machines [10]. It can be solved in pseudopolynomial time for a fixed number of machines by using either a dynamic program or a related shortest path formulation [15]. Here, we take the shortest path point of view. The nodes of the associated graph are tuples of the form $\langle k, w_1, \dots, w_m \rangle$; they are defined as follows: there is a source node $\langle 0, 0, \dots, 0 \rangle$. For each $k \in \{1, 2, \dots, n\}$ and $0 \le w_i \le b_i$ for $1 \le i \le m$, there is a node $\langle k, w_1, \dots, w_m \rangle$ denoting a schedule of the jobs $\{1, 2, \dots, k\}$ on the m machines such that the processing time on each machine i is exactly w_i. There is an arc from each node $\langle k, w_1, \dots, w_m \rangle$ to every node $\langle k + 1, w_1, \dots, w_{i-1}, w_i + p_{k+1}, w_{i+1}, \dots, w_m \rangle$, for $i \in \{1, 2, \dots, m\}$. The arc does not exist if $w_i + p_{k+1} > b_i$.

Observation 2.8. *There is a path from the source node* $\langle 0, 0, \dots, 0 \rangle$ *to some node* $\langle n, w_1, \dots, w_m \rangle$ *with* $w_i \le b_i$ *for each* i *if and only if the answer to the scheduling problem is "yes."*

Let $B := \max\{b_1, b_2, \dots, b_m\}$. The number of nodes and arcs of the network is $O(nB^m)$, which is pseudo-polynomial for fixed value of m. This running time is not polynomial for fixed or logarithmic precision without further modifications of the algorithm. We now show how the computational effort can be reduced to polynomial-time if the processing times are expressed with L-bit precision when L is either fixed or logarithmic. Let $p_j = a_j \cdot 2^{t_j}$, and assume that $a_j < 2^L$.

We first arrange the p_j's in non-increasing order of t_j's such that $t_1 \geq t_2 \geq \cdots \geq t_n$. Let $P_k := \sum_{j=k+1}^{n} p_j$. That is, P_k is the sum of the processing times of jobs $k+1, \ldots, n$.

Observation 2.9. *If $\langle k, w_1, \ldots, w_m \rangle$ is reachable from the source node $\langle 0, 0, \ldots, 0 \rangle$, then each w_i ($i = 1, \ldots, m$) is an integer multiple of 2^{t_k}, for $1 \leq k \leq n$.*

Observation 2.10. *If $\langle k, w_1, \ldots, w_m \rangle$ is reachable from the source node $\langle 0, 0, \ldots, 0 \rangle$ and if $w_i \leq b_i - P_k$ for some i, then the answer to the makespan problem is "yes."*

We refer to a node $\langle k, w_1, \ldots, w_m \rangle$ as a *terminal node* if $w_i \leq b_i$ for all i and $w_i \leq b_i - P_k$ for some i. We stop the algorithm at level k whenever it detects a terminal node of the form $\langle k, w_1, \ldots, w_m \rangle$ (and we delete all arcs emerging from such a node).

Lemma 2.11. *The number of nodes at level k that are reachable from $\langle 0, 0, \ldots, 0 \rangle$ is $\mathrm{O}(n^{m-1}2^{(m-1)L})$. The running time to determine these nodes is also $\mathrm{O}(n^{m-1}2^{(m-1)L})$, which is polynomial for fixed or logarithmic precision and a fixed number m of machines.*

Proof. If $\langle k, w_1, \ldots, w_m \rangle$ is reachable and not a terminal node, then $b_i - P_k < w_i \leq b_i$, and w_i is a multiple of 2^{t_k}. Therefore, there are at most $P_k/2^{t_k}$ such values, and this is bounded from above by $n2^L$. Thus, the number of reachable non-terminal nodes at level k is at most $(n2^L)^m$. This value is improved to $(n2^L)^{m-1}$ by observing that the value of the last coordinate is uniquely determined by the first $m-1$ coordinates. Finally, the number of reachable terminal nodes at level k is bounded by the number of reachable non-terminal nodes at level $k-1$. □

This gives an $\mathrm{O}(mn^{m-1}2^{(m-1)L})$ time algorithm for the recognition version of the makespan problem on m identical parallel machines when the processing times are expressed with L-bit precision. This is strongly polynomial time when m is fixed and L is fixed or logarithmic, i.e., under the fixed and logarithmic precision models. By using a common upper bound b on each machine (e.g., the value of Graham's greedy algorithm [12]) and choosing the reachable stage-n schedule with the lowest makespan, we obtain the following result.

Theorem 2.12. *The problem of minimizing the makespan on m identical parallel machines can be solved in $\mathrm{O}(mn^{m-1}2^{(m-1)L})$ time on m identical parallel machines when the processing times are expressed with L-bit precision. This is strongly polynomial-time when m is fixed and L is fixed or logarithmic, i.e., under the fixed and logarithmic precision models.*

Corollary 2.13. *The problem of partitioning n numbers into k parts of equal size, for a fixed k, is solvable in strongly polynomial time if the numbers are expressed with fixed or logarithmic precision.*

Proof. The problem of partitioning n numbers into k parts whose sum is equal is a special case of the makespan problem in which $k = m$ and $b_i = \sum_{j=1}^{n} p_j / k$ for all $1 \leq i \leq m$. □

2.3. The 3-partition problem

The 3-partition problem is defined as follows:

> Given $3n$ non-negative integers $c_1, c_2, \ldots, c_{3n}$ such that $\sum_{i=1}^{3n} c_i = nb$, is there a partition of the integers into n groups $S_1, S_2, \ldots, S_n$ of three elements each, such that $\sum_{c_i \in S_j} c_i = b$ for all $j = 1, 2, \ldots, n$?

This problem is strongly NP-complete [9]. This implies that the 3-partition problem is NP-complete under logarithmic precision. In this section, we develop a strongly polynomial-time algorithm for this problem under the fixed precision model.

For fixed L, suppose that the input integers are expressed with L-bit precision. Let $c_j = a_j \cdot 2^{t_j}$, where $a_j < 2^L$. We first establish an upper bound on the number of triples (z_1, z_2, z_3) whose sum is b, when each z_i is expressed with L-bit precision. Let $T_{L,b}$ be the set of triples of L-bit precision integers summing to b.

Lemma 2.14. *There are at most $(L+2)^2 2^{2L}$ triples of L-bit precision integers whose sum is b; i.e., the cardinality of the set $T_{L,b}$ is constant when L is fixed.*

Proof. Let $z_i = x_i \cdot 2^{y_i}$ for $1 \le i \le 3$ with $z_1 \ge z_2 \ge z_3$ and $z_1 + z_2 + z_3 = b$. Let $2^T \le b < 2^{T+1}$. Clearly, $y_1 \le T$. We also have $x_1 \cdot 2^{y_1} \ge b/3$. Hence, $x_1 \cdot 2^{y_1} > 2^{T-2}$ since $b \ge 2^T$. Therefore, $2^{y_1} > 2^{T-2}/2^L$ because $x_1 < 2^L$. Thus, $T - 1 - L \le y_1 \le T$, and the number of possible values for $z_1 = x_1 \cdot 2^{y_1}$ is bounded from above by $(L+2)2^L$. Once z_1 is fixed, it can be shown in a similar manner that there are at most $(L+2)2^L$ possible values for z_2. The values of z_1 and z_2 uniquely determine the value of z_3. Hence, the number of such triples (z_1, z_2, z_3) is at most $(L+2)^2 2^{2L}$. □

We now formulate the 3-partition problem under the L-bit precision model as an integer program with a fixed number of variables and $\mathrm{O}(n)$ constraints. In the integer programming formulation, there is a variable x_j for each triple j in $T_{L,b}$ that can be created from $c_1, c_2, \ldots, c_{3n}$. Let T be the subset of such triples. The variable x_j will denote the number of times that triple j appears in the solution. Lemma 2.14 shows that the number of variables x_j is constant.

Let $c'_1, c'_2, \ldots, c'_m$ $(m \le 3n)$ denote the distinct integers among $c_1, c_2, \ldots, c_{3n}$. Let n_i be the number of times that c'_i appears among the input integers for $1 \le i \le m$. Let α_{ij} $(0 \le \alpha_{ij} \le 3)$ be the number of times that c'_i appears in triple j. Let T_i be the set of triples from T that contain c'_i. We require a non-negative, feasible solution to an integer program with the following m constraints:

$$\sum_{j \in T_i} \alpha_{ij} x_j = n_i \quad \text{for } 1 \le i \le m.$$

Such an integer program can be solved in time polynomial in n due to the fact that the number of variables is constant and the number of constraints is $\mathrm{O}(n)$ [18]. This leads to a strongly polynomial-time algorithm for the 3-partition problem.

Theorem 2.15. *The 3-partition problem can be solved in strongly polynomial time under the fixed precision model, i.e., when the input integers are expressed with L-bit precision for fixed L.*

3. L-bit precision, ε-optimization schemes, and bit rounding

In this section, we make the transition from computing in the L-bit precision model to designing ε-optimization schemes. One connection is the following. Suppose one wants a solution to $\min\{cx : x \in X\}$ that is accurate within a factor of $\varepsilon = 1/2^L$; i.e., one that is $1/2^L$-optimal. Then one can round each (integer) cost coefficient c_j and express it using just $L+1$ bits of accuracy. If we solve the rounded problem optimally, the optimal solution x^* for the rounded problem is ε-optimal. Indeed, consider an arbitrary index j, and assume that $2^K \le c_j < 2^{K+1}$. (A similar argument works for negative coefficients.) We can represent c_j as $c_j = \sum_{i=0}^{K} c_j^i 2^i$ with $c_j^i \in \{0, 1\}$ for all $i = 0, 1, \ldots, K$. Just keeping the leading $L+1$ bits yields $c'_j = \sum_{i=K-L}^{K} c_j^i 2^i \ge c_j - 2^{K-L} \ge c_j(1 - 1/2^L)$. The result follows.

We now consider the relation between ε-optimization and what we called bit rounding in the introduction. Recall that in the bit rounding problem, the cost coefficients are given with L-bit precision, and we have to determine an optimal solution, but we are granted the freedom of choosing the $(L+1)$st bit.

Theorem 3.1. *Let Π be a combinatorial optimization problem such that, for any instance (c, X) of Π, (c', X) is also an instance of Π, for any vector c' of suitable dimension. If Π has an FPTEOS, then the bit rounding version of Π can be solved in polynomial time under logarithmic precision; if Π has a PTEOS, its bit rounding problem can be solved in polynomial time under fixed precision.*

Proof. Consider an instance of the bit rounding version of Π with costs $c_j = a_j \cdot 2^{t_j}$ and $|a_j| < 2^L$ for all j. To simplify the notation, we assume that $c_j \ge 0$. We need to find a feasible solution x^* that is optimal with respect to costs c' where $c'_j = 2a_j \cdot 2^{t_j}$ or $c'_j = (2a_j + 1) \cdot 2^{t_j}$.

Let $\bar{c}_j = (2a_j + 1/2) \cdot 2^{t_j}$ for all j. Run the given (F)PTEOS for problem Π for cost vector $\bar{c}$ and $\varepsilon = 1/2^{L+2}$. This gives a feasible solution x^* and a cost vector $\tilde{c}$ such that x^* is optimal for $\tilde{c}$ and $(1-\varepsilon)\bar{c}_j \le \tilde{c}_j \le (1+\varepsilon)\bar{c}_j$. Note that $4a_j + 1 \le 2^{L+2} = 1/\varepsilon$, or $\varepsilon(2a_j + 1/2) \le 1/2$. Hence, $2a_j \cdot 2^{t_j} \le \tilde{c}_j \le (2a_j + 1) \cdot 2^{t_j}$.

We obtain the cost vector c' as follows: if $x_j^* = 1$, then $c'_j = 2a_j \cdot 2^{t_j}$ (round down $\tilde{c}_j$), else if $x_j^* = 0$, then $c'_j = (2a_j + 1) \cdot 2^{t_j}$ (round up $\tilde{c}_j$). The feasible solution x^* and the cost vector c' is the answer to the original bit rounding version of problem Π.

```
Initialization:
  Let $\delta := \varepsilon/2n$.
  Let $I := J := \emptyset$; $K := \{1, 2, \dots, n\}$.
  Let $x^*$ be any feasible solution.

Main Loop:
  while $K \neq \emptyset$ do
    Find a $\delta$-approximate completion $x'$ of $\langle I, J \rangle$.
    if $cx' < cx^*$, then $x^* := x'$.
    Let $U := \max\{c_k : k \in K\}$.
    for all indices $k \in K$ with $c_k > U/2$, do
      if $x^*_k = 1$, then $J := J \cup \{k\}$, and $K := K \setminus \{k\}$;
      if $x^*_k = 0$, then $I := I \cup \{k\}$, and $K := K \setminus \{k\}$.
```

Fig. 1. Converting an FPTAS into an FPTEOS.

To prove the optimality of x^* for cost vector c', we need to show that $c'x^* \leq c'x$ for any feasible solution x. Because of the way in which c' is obtained using $\tilde{c}$ and x^*, it is easy to see that $(c'_j - \tilde{c}_j)(x^*_j - x_j) \leq 0$ for all j, i.e., $(c' - \tilde{c})(x^* - x) \leq 0$. Also, because x^* is optimal for $\tilde{c}$, we have $\tilde{c}(x^* - x) \leq 0$. Hence, $c'(x^* - x) = (c' - \tilde{c})(x^* - x) + \tilde{c}(x^* - x) \leq 0$.

Finally, note that if (A_ε) is an FPTEOS for problem Π, then A_ε has running time polynomial in $1/\varepsilon$, and the above procedure is polynomial-time under logarithmic precision ($1/\varepsilon = \mathrm{O}(\mathrm{poly}(n))$ since $L = \mathrm{O}(\log n)$). Otherwise, if (A_ε) is a PTEOS for Π, then the above procedure is polynomial-time under fixed precision ($1/\varepsilon = \mathrm{O}(1)$ since $L = \mathrm{O}(1)$). □

4. Fully polynomial-time *ε*-optimization schemes

In this section, we explore the relationship between ordinary fully polynomial-time approximation schemes and fully polynomial-time ε-optimization schemes. We start by observing that the existence of the latter implies that of the former:

Lemma 4.1. *Let* $\min\{cx : x \in X\}$ *be a minimization problem with non-negative cost coefficients* ($c \geq 0$) *and non-negative solutions (i.e., $x \in X$ implies $x \geq 0$). If the solution $x' \in X$ is ε-optimal for some* $0 < \varepsilon < 1$*, then its relative error is at most* $2\varepsilon/(1 - \varepsilon)$.

Proof. Let $x \in X$ be an arbitrary feasible solution, and let $x' \in X$ be ε-optimal. There exists an ε-perturbation c' such that x' is optimal for $\min\{c'x : x \in X\}$. Hence,

$$cx' \leq \frac{1}{1-\varepsilon}c'x' \leq \frac{1}{1-\varepsilon}c'x \leq \frac{1+\varepsilon}{1-\varepsilon}cx.$$

The claim follows. □

We now work towards establishing a partially inverse result. A *partial solution* for a combinatorial optimization problem $\min\{cx : x \in X\}$ with $X \subseteq \{0, 1\}^n$ is a pair $\langle I, J \rangle$, where I and J are disjoint subsets of indices; i.e., $I \cup J \subseteq \{1, \dots, n\}$ and $I \cap J = \emptyset$. Let $K := \{1, \dots, n\} \setminus (I \cup J)$. A *completion* of a partial solution $\langle I, J \rangle$ is a feasible solution x such that $x_j = 0$ for $j \in I$, and $x_j = 1$ for $j \in J$. A δ-*approximate completion* of a partial solution $\langle I, J \rangle$ is a completion x such that for any other completion y we have $\sum_{j \in K} c_j x_j \leq (1 + \delta) \sum_{j \in K} c_j y_j$.

Suppose that we have an FPTAS that can also deliver a δ-approximate completion of any partial solution. We claim that we can use this FPTAS to determine a solution that is ε-optimal in time that is polynomial in the size of the problem and in $1/\varepsilon$.

Theorem 4.2. *Let Π be a combinatorial optimization problem with non-negative cost coefficients and an FPTAS that also works for any instance that arises from a given instance of Π by fixing some of the variables. Then, the algorithm in Fig. 1 is an FPTEOS for Π.*

A prototypical example to which Theorem 4.2 can be applied is the knapsack problem. Indeed, any partial fixing of variables gives rise to a new knapsack problem.

Proof. Let x^* be the solution returned by the algorithm. Let $c^*_j = (1+\varepsilon)c_j$ if $x^*_j = 0$, and let $c^*_j = (1-\varepsilon)c_j$ if $x^*_j = 1$. We claim that x^* is optimal for the original problem if c is replaced by c^*.

Let x be any feasible solution, $x \neq x^*$. We will show that $c^*x^* \leq c^*x$. Among all indices k such that $x^*_k \neq x_k$, let r be the one for which c_r is maximum. Let $\langle I, J\rangle$ be the partial solution at the beginning of the main loop prior to x^*_r being fixed. Let $K = \{1, \ldots, n\} \setminus (I \cup J)$.

Note that x^* is a δ-approximate completion of $\langle I, J\rangle$. Moreover, $x_k = x^*_k$ for all $k \in I \cup J$, i.e., x is a completion of $\langle I, J\rangle$, too. For if not, then $x_s \neq x^*_s$ for some $s \in I \cup J$. Since x^*_s was fixed in an iteration earlier than x^*_r, it follows that $c_s > c_r$. This contradicts the maximality of c_r. Hence,

$$\sum_{j\in K} c_j(x^*_j - x_j) \leq \delta \sum_{j\in K} c_j \leq \delta|K|2c_r \leq \varepsilon c_r.$$

Moreover, by the definition of c^* we have $(c^*_j - c_j)(x^*_j - x_j) \leq 0$ for each $j \in K$, and $(c^*_r - c_r)(x^*_r - x_r) = -\varepsilon c_r$. Therefore,

$$c^*(x^* - x) = \sum_{j\in K} c^*_j(x^*_j - x_j) = \sum_{j\in K} c_j(x^*_j - x_j) + \sum_{j\in K}(c^*_j - c_j)(x^*_j - x_j) \leq 0. \quad \square$$

5. (Fully) polynomial-time ε-optimization schemes for specific combinatorial optimization problems

We now turn to the design of ε-optimization schemes for specific problems. We present two techniques to obtain such schemes. We illustrate them with the help of a scheduling with rejection problem and the bin packing problem, respectively.

5.1. *Single-machine scheduling with rejection*

The first technique resembles the previously presented way of turning an FPTAS into an FPTEOS. But let us first describe the problem setting. In most scheduling problems in the literature, one ignores the possibility of outsourcing a task. However, in practice, it is often possible to outsource a task, and have it completed by another manufacturer. One can model this as scheduling with rejection, see, e.g., [8,21]. If a job j is rejected, it means that we do not need to process it, but we incur a cost e_j for rejecting it. The cost e_j reflects the cost of having to outsource task j. Suppose that job j has a processing time of p_j and a due date d_j. Let C_j denote the completion time of the jth job, assuming that it is not rejected. Let L_j denote the lateness of the jth job. Then $L_j = C_j - d_j$, and it may be negative. Let S denote the set of jobs that are scheduled. Let R denote the set of jobs that are rejected. The objective function is to minimize $f(R, S) := \sum_{k\in R} e_k + \max_{k\in S} L_k$, which can be negative. An NP-completeness proof can be found in [21]. We can represent a solution as (R, S) with the understanding that jobs in S are scheduled in non-decreasing order of their due dates, which is optimal for single-machine scheduling to minimize maximum lateness [17]. Of course, $R \cap S = \emptyset$ and $R \cup S = \{1, 2, \ldots, n\}$.

We let SCHEDULE (p, e, d, R^*, S^*) be a subroutine that finds the optimal schedule if the jobs have processing times p_j, rejection costs e_j, due dates d_j, the jobs in R^* must be rejected, and the jobs in S^* must be scheduled ($R^* \cap S^* = \emptyset$ and $R^* \cup S^* \subseteq \{1, 2, \ldots, n\}$). In fact, [21] contains an algorithm that is pseudopolynomial in $\sum_{j=1}^n e_j$ and does just that. We repeat it here for completeness. For simplicity, we assume that $R^* = S^* = \emptyset$; the extension to general sets is straightforward. In a first step, we design a dynamic program that produces a schedule which minimizes the maximum lateness when the total rejection penalty γ of the rejected jobs is prescribed. In a second step, we iteratively call this dynamic program for all possible values of γ. For the dynamic program, we assume that jobs are numbered in non-decreasing order of their dues dates d_j; i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$. Let $\phi_j(\gamma)$ be the minimum value of the maximum lateness objective when the jobs in consideration are $j, j+1, \ldots, n$. Then,

$$\phi_j(\gamma) = \begin{cases} \max\{\phi_{j+1}(\gamma) + p_j, p_j - d_j\} & \text{if } j < n \text{ and } \gamma < e_j, \\ \min\{\phi_{j+1}(\gamma - e_j), \max\{\phi_{j+1}(\gamma) + p_j, p_j - d_j\}\} & \text{if } j < n \text{ and } \gamma \geq e_j, \\ -\infty & \text{if } j = n \text{ and } \gamma = e_n, \\ p_n - d_n & \text{if } j = n \text{ and } \gamma = 0, \\ \infty & \text{otherwise.} \end{cases}$$

```
$R^* := S^* := \emptyset$.
Let $(R, S)$ be an arbitrary feasible solution.
Let $e_1 \geq e_2 \geq \cdots \geq e_n$.
for $j = 1$ to $n$ do
    let $q := e_j \cdot \varepsilon / n$;
    for $i = j$ to $n$, $e'_i := q \cdot \lceil e_i / q \rceil$;
    $(R', S') :=$ SCHEDULE$(p, e', d, R^*, S^*)$;
    if $f(R', S') \leq f(R, S)$, then $(R, S) := (R', S')$;
    if $j \in S$, then $S^* := S^* \cup \{j\}$
        else if $j \in R$, then $R^* := R^* \cup \{j\}$.
for each $i \in R^*$, $e^*_i := e_i \cdot (1 - \varepsilon)$.
for each $i \in S^*$, $e^*_i := e_i \cdot (1 + \varepsilon)$.
return $(R^*, S^*)$, $e^*$.
```

Fig. 2. An FPTEOS for scheduling with rejection.

Eventually, we find a schedule that minimizes $f(R, S)$ by calling this dynamic program $\sum_{j=1}^{n} e_j + 1$ times:

$$\min \left\{ \phi_1(\gamma) + \gamma : 0 \leq \gamma \leq \sum_{j=1}^{n} e_j \right\}.$$

We will write $(R, S) :=$ SCHEDULE (p, e, d, R^*, S^*) to indicate that S is the set of scheduled jobs output by the subroutine SCHEDULE, and R is the set of rejected jobs. Obviously, $S^* \subseteq S$ and $R^* \subseteq R$. The algorithm described in Fig. 2 produces an ε-optimal solution.

Theorem 5.1. *Let (R^*, S^*) and e^* be the output of the algorithm presented in Fig. 2. Then, (R^*, S^*) is an optimal solution with respect to processing times p_j, due dates d_j, and rejection costs e^*_j. The total running time of the algorithm is $O(n^4/\varepsilon)$, and hence it is an FPTEOS for scheduling with rejection to minimize maximum lateness.*

Proof. As in the algorithm, we assume that $e_1 \geq e_2 \geq \cdots \geq e_n$. We denote by (R'_j, S'_j) the output (R', S') of SCHEDULE in the jth iteration of the for-loop. Let (R_j, S_j) be the schedule (R, S) after the jth iteration of the for-loop. Because (R, S) is not changed if the objective value of the new schedule (R', S') is worse, we have $f(R^*, S^*) = f(R_n, S_n) \leq f(R_{n-1}, S_{n-1}) \leq \cdots \leq f(R_1, S_1)$. Moreover, $f(R^*, S^*) \leq f(R'_j, S'_j)$ for all j.

Let f^* denote the objective function when the rejection cost vector is replaced by e^*. Similarly, we will use f' when the rejection cost vector is e'. Let (R, S) be an arbitrary feasible solution that is different from (R^*, S^*). We have to show that $f^*(R^*, S^*) \leq f^*(R, S)$.

Let r be the smallest index for which (R^*, S^*) differs from (R, S). We claim that

$$f^*(R^*, S^*) - f^*(R, S) \leq f(R^*, S^*) - f(R, S) - \varepsilon e_r. \tag{5.1}$$

In fact, let us rewrite this inequality as $f^*(R^*, S^*) - f(R^*, S^*) \leq f^*(R, S) - f(R, S) - \varepsilon e_r$. We consider two cases. If $r \in R \cap S^*$, then the perturbation e^* increases the cost of rejecting job r in (R, S), which implies the inequality. On the other hand, if $r \in S \cap R^*$, the perturbation e^* decreases the cost of rejecting job r in (R^*, S^*); again, the inequality follows.

Let e' be the rejection cost vector used for the call of SCHEDULE in iteration r. We have

$$f(R^*, S^*) \leq f(R'_r, S'_r) \leq f'(R'_r, S'_r) \leq f'(R, S) \leq f(R, S) + \varepsilon e_r. \tag{5.2}$$

Here, the first inequality follows from prior observations. The second one is implied by the definition of e'. The third inequality holds because $R'_r \cap \{1, 2, \ldots, r-1\} = R \cap \{1, 2, \ldots, r-1\}$, $S'_r \cap \{1, 2, \ldots, r-1\} = S \cap \{1, 2, \ldots, r-1\}$, and (R'_r, S'_r) is optimal with respect to e' when the assignment of jobs $\{1, 2, \ldots, r-1\}$ is fixed accordingly. Eventually, the last inequality follows from the definition of e' and q in the rth iteration of the algorithm. Adding up inequalities (5.1) and (5.2) yields $f^*(R^*, S^*) - f^*(R, S) \leq 0$.

Let us finally analyze the running time of this algorithm. Any time SCHEDULE is called with an objective function e', each coefficient of e' is a multiple of q. Hence, we only need to compute table entries $\phi_j(\gamma)$ for γ being a multiple of q. There are at most $O(n^3/\varepsilon)$ entries of this kind, each of which can be computed itself in $O(1)$ time. Hence the overall running time of the FPTEOS is $O(n^4/\varepsilon)$. □

5.2. *Bin packing*

The second technique to produce an ε-optimization scheme is somewhat more problem-specific. Moreover, instead of changing the objective function coefficients, we modify other parameters of the problem. We consider the bin packing problem, which does not have an ordinary PTAS (unless P = NP). Asymptotic approximation schemes were proposed in [7,19]. We are given n items for packing; each bin has size 1, and we want to minimize the number of bins needed to fit all items. In our context, we are also given $0 < \varepsilon \leq 1/2$.

In the first step of the algorithm, we partition the items into two groups, group A with items of size smaller than $\varepsilon/3$, and the remaining items are in group B. In the second step, we round the elements in group B so that their sizes are an element of the set $R := \{\varepsilon/3 \cdot (1+\varepsilon/2)^k \,:\, \text{for some integer } k\}$. This creates a rounding in which items are rounded by at most a factor of $(1+\varepsilon/2)$. Let $q(\varepsilon)$ be the number of different values in R smaller than 1; for fixed ε, the value of $q(\varepsilon)$ is a constant.

In the third step, we find the optimum packing of the rounded elements in group B. For this, we use integer programming. There are at most $3/\varepsilon$ elements in any bin. Furthermore, there are at most $q(\varepsilon)$ different sizes in R. Consequently, for a fixed value of ε, there is just a constant number of different ways of packing a bin. Let P be the set of all such packing patterns, and let x_p be the number of bins packed according to pattern $p \in P$. One then formulates this part of the bin packing problem as an integer program with variables x_p. The objective function is $\sum_{p \in P} x_p$ and there is one constraint for each item size in R. This integer program is solvable in $O(L)$ time, where L is the number of bits in the largest integer [18]. Suppose that the optimal solution for the integer program uses K bins.

In the fourth and final step, we greedily pack the items from group A. We either end up with a packing with K bins (in which case we are optimal for the perturbed problem) or else we need more than K bins. In the latter case, we can round the sizes again so that all but one bin is full, in which case we are optimal for the reperturbed problem. In fact, consider a bin that is not full, but cannot fit any of the remaining items in A. In particular, the total size of the items already contained in this bin is at least $1 - \epsilon/3$. Hence, we need to multiply each item size by at most $1 + \varepsilon/(3-\varepsilon)$ to obtain a full bin. Note that while small items were not rounded before, we have $(1+\varepsilon/2)(1+\varepsilon/(3-\varepsilon)) \leq 1+\varepsilon$ for big items.

A similar reasoning can be applied to the strongly NP-hard problem of scheduling jobs on identical parallel machines so as to minimize the makespan. Adapting the PTAS of Hochbaum and Shmoys [14] accordingly, we obtain a PTEOS by perturbing job processing times. Hence, we have the following theorem.

Theorem 5.2. *Both the bin packing problem and the identical parallel machine scheduling problem to minimize the makespan have a PTEOS.*

More generally, one can observe that whenever an FPTAS or a PTAS is obtained by the technique of "rounding the input" [23], then there is a good chance that this technique can also be used to generate an FPTEOS or a PTEOS, respectively.

6. Concluding remarks

The standard notion of measuring the complexity of an algorithm or problem can fail in situations in which some or all of the input data may consist of numbers of different orders of magnitude, each of which has relatively few relevant bits only. The L-bit precision model that we propose here attempts to address this issue. Several otherwise NP-hard problems have algorithms that run in polynomial time, assuming a compact encoding of the data that are given to us with L bits of accuracy only. Here, L is fixed or logarithmic in the number of data items.

Similarly, if the objective function coefficients of a combinatorial optimization problem differ drastically from one another in terms of magnitude, conventional approximation schemes essentially "round" small integer coefficients to zero. In situations in which this effect is unwanted, ε-optimization schemes provide an alternative. Instead of relying on the concept of relative error, they measure the distance from optimality in terms of the amount of perturbation to the objective function that is needed to make the resulting solution optimal. As a consequence, they also apply to situations in which no ordinary approximation schemes can exist, such as for the 3-partition problem as well as to problems in which the objective can take on positive or negative values.

In situations in which the cost data are only known approximately (as is common), the ε-optimization schemes make practical sense. They find an optimal solution for a problem that has the same feasible region and approximately the same costs as the original problem.

We hope that the concepts and algorithmic schemes proposed here may enhance the toolbox of the algorithm designer when it comes to addressing certain situations in which some of the more conventional methods may fall short.

Acknowledgments

This work was supported by the Office of Naval Research under contract ONR N00014-98-1-0317. The authors are also grateful to the two anonymous referees whose comments helped to improve the presentation of this paper.

References

[1] R. Ahuja, T. Magnanti, J. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
[2] R. Ahuja, J. Orlin, Inverse optimization, Operations Research 49 (2001) 771–783.
[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties, Springer, 1999.
[4] R. Bellman, Dynamic Programming, Princeton University Press, 1957.
[5] W. Cody, J. Coonen, D. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. Ris, D. Stevenson, A proposed radix- and word-length-independent standard for floating point arithmetic, IEEE Micro 4 (1984) 86–100.
[6] G. Dantzig, Discrete-variable extremum problems, Operations Research 5 (1957) 266–277.
[7] W. Fernandez de la Vega, G. Lueker, Bin packing can be solved within $1 + \varepsilon$ in linear time, Combinatorica 1 (1981) 349–355.
[8] D. Engels, D. Karger, S. Kolliopoulos, S. Sengupta, R. Uma, J. Wein, Techniques for scheduling with rejection, Journal of Algorithms 49 (2003) 175–191.
[9] M. Garey, D. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM Journal on Computing 4 (1975) 397–411.
[10] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[11] D. Goldberg, What every computer scientist should know about floating-point arithmetic, Computing Surveys 23 (1991) 5–48.
[12] R. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal of Applied Mathematics 17 (1969) 263–269.
[13] D. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS, 1997.
[14] D. Hochbaum, D. Shmoys, Using dual approximation algorithms for scheduling problems: Theoretical and practical results, Journal of the ACM 34 (1987) 144–162.
[15] E. Horowitz, S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, Journal of the ACM 23 (1976) 317–327.
[16] IEEE, IEEE standard for binary floating-point arithmetic, SIGPLAN Notices 22 (1985) 9–25.
[17] J. Jackson, Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.
[18] H. Lenstra Jr., Integer programming with a fixed number of variables, Mathematics of Operations Research 8 (1983) 538–548.
[19] N. Karmarkar, R. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, in: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, IEEE, 1982, pp. 312–320.
[20] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.
[21] S. Sengupta, Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection, in: F. Dehne, J. Sack, M. Smid (Eds.), Algorithms and Data Structures, in: Lecture Notes in Computer Science, vol. 2748, Springer, 2003, pp. 79–90.
[22] V. Vazirani, Approximation Algorithms, Springer, 2001.
[23] G. Woeginger, When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing 12 (2000) 57–75.