An approximate algorithm for combinatorial optimization problems with two parameters

David Blokh, * Gregory Gutin †

Abstract

We call a *minimum cost restricted time combinatorial optimization* (MCRT) problem any problem that has a finite set P, finite family $\mathcal{S}$ of subsets of P, non-negative threshold h, and two non-negative real-valued functions $y : P \rightarrow \mathbf{R}_+$ (say, cost) and $x : P \rightarrow \mathbf{R}_+$ (say, time). One seeks a solution $F^* \in \mathcal{S}$ with $y(F^*) = \min\{y(F) : F \in \mathcal{S}, x(F) \leq h\}$, where $z(G) = \sum_{g \in G} z(g)$ for every $z \in \{x, y\}$ and $G \in \mathcal{S}$. We also assume that for the corresponding minimum cost problem there is an efficient exact or approximate algorithm. We describe a very simple approximate algorithm for any MCRT problem. Though our algorithm is not polynomial in general, we provide some evidence that the algorithm may be fairly fast in many cases.

1 Introduction

It is well known that there is a large variety of different real-world optimization problems (cf. [2, 8, 14]). Thus, along with development of very effective algorithms solving some important optimization problems, it seems to be a right strategy to design algorithms for quite general optimization problems. In this paper, we propose a very simple algorithm for a wide family of different combinatorial optimization problems with two parameters. Problems

*Dept. of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel.

†Corresponding author. Dept. of Maths and Stats, Brunel University, Uxbridge, Middlesex UB8 3PH, U.K. and Dept. of Math. and Compt. Sci., Odense University, DK-5230, Odense, Denmark.

from this family are often appear in both theory and practice of combinatorial optimization (cf. [1, 3, 4, 5, 6, 9, 11, 12, 13, 15], etc.) Though our algorithm is not polynomial in general (which seems to be impossible to expect in such general setting), we provide some evidence that the algorithm may be fairly fast in many cases.

We call a *minimum cost restricted time combinatorial optimization* (MCRT) problem any problem that has a finite set P, finite family $\mathcal{S}$ of subsets of P, non-negative threshold h, and two non-negative real-valued functions $y : P \to \mathbf{R}_+$ (say, cost) and $x : P \to \mathbf{R}_+$ (say, time). One seeks a solution $F^* \in \mathcal{S}$ with $y(F^*) = \min\{y(F) : \ F \in \mathcal{S}, x(F) \leq h\}$, where $x(G) = \sum_{g \in G} x(g)$, $y(G) = \sum_{g \in G} y(g)$ and $G \in \mathcal{S}$. We also assume that for the corresponding minimum cost problem there is an efficient exact or approximate algorithm. For simplicity, in what follows, we assume that the last algorithm is exact.

A well known example of a MCRT problem is the restricted shortest path (RSP) problem. This problem is NP-hard [2]. Some authors constructed practical non-polynomial exact algorithms for the RSP problem (cf. [1, 4, 6, 7, 8]). Others developed fully polynomial approximation schemes (cf. [5, 12, 15]). Another example of a MCRT problem is the two parameters minimum spanning tree problem. In [3, 11], polynomial approximate algorithms for this problem were constructed. The two parameters knapsack and traveling salesman problem were considered in [13] and [9], respectively. Note that the polynomial approximate algorithms from [3, 5, 11, 12, 15] utilize special properties of the corresponding one parameter problems.

In this paper, we describe an approximate algorithm for any MCRT problem. One of the advantages of our algorithm is its simplicity. This turns out to be an important property for practical algorithms since more sophisticated algorithms may not be faster in practice (cf. [10]).

We obtain a bound for the performance ratio of a found solution that depends on both instance of a MCRT problem and solution. It is difficult to expect a practical approximate algorithm with satisfactory guaranteed performance ratio for any MCRT problem. Moreover, an a posteriori performance ratio may better evaluate the real performance ratio of the found solution than a "uniform" guaranteed one.

2 Description of the algorithm

We first provide an informal description of our algorithm. We have assumed that there is an effective algorithm $\mathcal{A}$ for the corresponding minimum cost problem. Using $\mathcal{A}$ we can find two solution F and H such that $y(F) = \min\{y(T) : T \in \mathcal{S}\}$ and $x(H) = \min\{x(T) : T \in \mathcal{S}\}$. If $x(F) \leq h$, then F is an optimal solution. If $x(H) > h$, then there is no solutions for our problem. Therefore, we may assume that $x(H) \leq h < x(F)$. The main idea of the algorithm is to apply $\mathcal{A}$ to a "linear combination" of F and H in order to obtain a feasible solution which is better than H. Represent F and H as points with coordinates $(x(F), y(F))$ and $(x(H), y(H))$ in a rectangular Cartesian coordinate system. We shall use the equation of the straight line FH in the form $ax + by = c$, where $a = y(H) - y(F)$, $b = x(F) - x(H)$ and $c = x(F)y(H) - x(H)y(F)$. Associate the new weight $w(p) = ax(p) + by(p)$ to each element $p \in P$ and, using $\mathcal{A}$, find $G \in \mathcal{S}$, a solution to the problem $\min\{w(T) : T \in \mathcal{S}\}$. If G lies on the line FH, then the approximate solution is either G if G is feasible, i.e. $x(G) \leq h$, or H if $x(G) > h$. If G does not belong to FH, then, it is easy to see, that $y(G) < y(H)$. If G is feasible, we replace H by G, otherwise - F by G, and repeat our iteration with the new pair H, F.

We now give a more formal description of our algorithm.

Step 1. Using $\mathcal{A}$ find F so that $y(F) = \min\{y(T) : T \in \mathcal{S}\}$. If $x(F) \leq h$, then F is an optimal solution. Stop.

Step 2. Using $\mathcal{A}$ find H so that $x(H) = \min\{x(T) : T \in \mathcal{S}\}$. If $x(H) > h$, then there is no solution. Stop.

Step 3. Set $a := y(H) - y(F)$, $b := x(F) - x(H)$ and $c := x(F)y(H) - x(H)y(F)$. Compute $w(p) := ax(p) + by(p)$ for every $p \in P$. Using $\mathcal{A}$ find G so that $w(G) = \min\{w(T) : T \in \mathcal{S}\}$.

Step 4. If $c = ax(G) + by(G)$, then G is a solution when $x(G) \leq h$ and H is a solution, otherwise (i.e. $x(G) > h$). Stop.

Step 5. If $c > ax(G) + by(G)$, then set $H := G$ when $x(G) \leq h$ and $F := G$, otherwise. Go to Step 3.

3 Performance of the algorithm

In this section, we obtain upper bounds for the time complexity of our algorithm and its performance ratio.

Let F, H be arbitrary solutions from $\mathcal{S}$ represented as points with coordinates $(x(F), y(F))$ and $(x(H), y(H))$ in a Cartesian coordinate system and let R be another point on the plane.

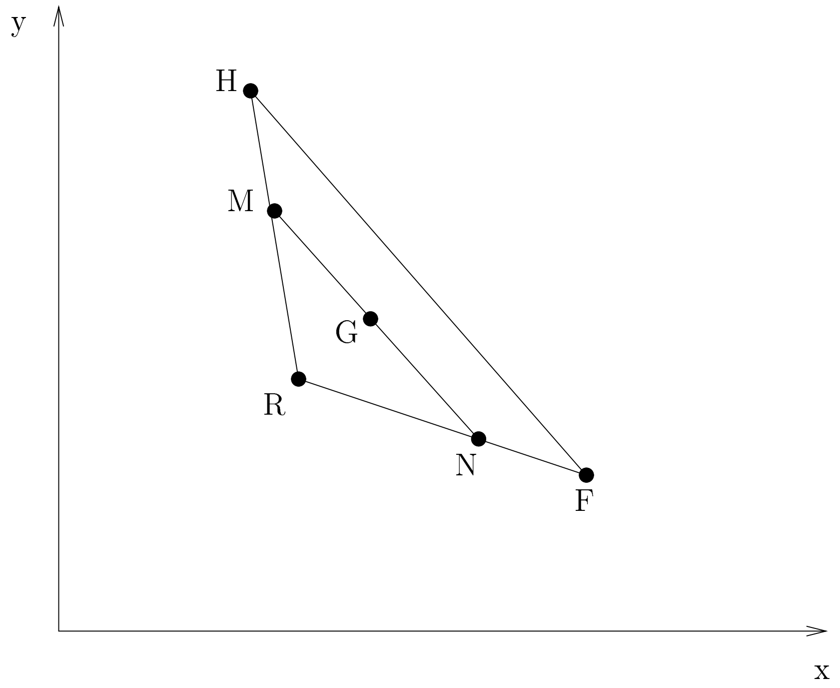

Fig. 1

Lemma 3.1 *Suppose that $x(H) \leq x(R) \leq x(F)$, $y(F) \leq y(R) \leq y(H)$ and $ax(R) + by(R) < c$, where a, b and c are defined as in Step 3 of the algorithm. Let G, $G \neq R$, be the solution found in Step 3 of the algorithm given F and H above and let the line that is parallel to the line FH and contains G intersect the lines HR and FR in the points M and N, respectively (see Fig. 1). Then*

$$x(H) \leq x(M) \leq x(G) \leq x(N) \leq x(F), \tag{1}$$
$$y(F) \leq y(N) \leq y(G) \leq y(M) \leq y(H). \tag{2}$$

Proof: Since M and N are points belonging to the segments $[HR]$ and $[FR]$, respectively, and $x(H) \leq x(R) \leq x(F)$, the x-coordinates of M and N satisfy the inequalities $x(H) \leq x(M)$, $x(N) \leq x(F)$. G lies on the segment $[MN]$. Therefore, $x(M) \leq x(G) \leq x(N)$. Thus, (1) holds. Analogously, we can prove (2). $\square$.

Theorem 3.2 *After every performance of Step 5 the value of $x(F) + y(H)$ decreases by positive number.*

Proof:

Suppose that Step 3 of the algorithm is performed $\tau + 1$ times, $\tau \geq 2$. Let F_i, H_i and G_i be the current values of F, H and G in the i-th performance of Step 3 ($i = 1, 2, ..., \tau$). R_i, M_i and N_i are defined recursively as follows. R_1 denotes the point with coordinates $x(R_1) = x(H_1)$ and $y(R_1) = y(F_1)$. For $i = 1, 2, ..., \tau$, M_i and N_i denote the points where the line that is parallel to the line F_iH_i and contains G_i intersects the lines H_iR_i and F_iR_i. For $i = 2, 3, ..., \tau$, $R_i = M_{i-1}$ if $x(G_{i-1}) > h$ and $R_i = N_{i-1}$ if $x(G_{i-1}) \leq h$.

We first prove by induction on i that

$$x(H_i) \leq x(M_i) \leq x(G_i) \leq x(N_i) \leq x(F_i), \tag{3}$$

$$y(F_i) \leq y(N_i) \leq y(G_i) \leq y(M_i) \leq y(H_i) \tag{4}$$

for every $i = 1, 2, ..., \tau$.

The case $i = 1$ follows from Lemma 3.1 and the definition of R_1. Let $i \geq 2$. Then (3) and (4) follow from the definition of R_i, induction hypothesis, and Lemma 3.1.

To prove the theorem, it is sufficient to show that for every $i = 1, 2, ..., \tau$

$$\text{if } x(G_i) \leq h, \text{ then } y(G_i) < y(G_j), \text{ where } j = \max\{k : -1 \leq k < i, x(G_k) \leq h\}, \tag{5}$$

and

$$\text{if } x(G_j) > h, \text{ then } x(G_i) < x(G_m), \text{ where } m = \max\{k : 0 \leq k < i, x(G_k) > h\}, \tag{6}$$

where $G_{-1} = H_1$ and $G_0 = F_1$.

Clearly, $H_i = G_j$ and $F_i = G_m$, where j and m are defined in (5) and (6). Assume that $x(G_i) > h$, but $x(G_i) \geq x(G_m) = x(F_i)$. By (4), $y(G_i) \geq y(F_i)$. Thus, $ax(G_i) + by(G_i) \geq ax(F_i) + by(F_i)$. However, this is impossible since $i \leq \tau$. So, (6) has been proved. The claim (5) can be shown analogously. $\Box$.

A point (x', y') is an $\mathcal{S}$-*point* if there is $C \in \mathcal{S}$ such that $x' = x(\mathcal{S})$, $y' = y(\mathcal{S})$. An $\mathcal{S}$-point is *minimal* if there is no $\mathcal{S}$-point (x'', y'') such that $x'' \leq x', y'' \leq y'$ and $x'' + y'' < x' + y'$. Let $m(\mathcal{S})$ denote the number of minimal $\mathcal{S}$-points. Obviously, for every solution G obtained in Step 3, $(x(G), y(G))$ is a minimal $\mathcal{S}$-point. It follows from Theorem 3.2 that after Step 5 we obtain distinct elements G of $\mathcal{S}$.

Corollary 3.3 *1) Step 3 of the algorithm is performed at most* $m(\mathcal{S}) + 1$ *times.*

2) The algorithm requires $O(m(\mathcal{S})(|P| + t(\mathcal{A})))$ *time, where* $t(\mathcal{A})$ *is the time complexity of* $\mathcal{A}$.

If the functions x and y are integer valued, then we can obtain a simple dynamic upper bound for the number of iterations of Step 3 to be still performed (this upper bound might be of interest for interactive implementations of the algorithm). By Theorem 3.2, the value of $x(F) + y(H)$ decreases by at least one after every performance of Step 5. Since every G is a minimal $\mathcal{S}$-point, each of the functions $y(H) - y(F)$ and $x(F) - x(H)$ decreases by at least one.

Corollary 3.4 *Let* F_0 *and* H_0 *be the current values of points* F *and* H *in an iteration of Step 3 of the algorithm. Then the number of iterations of Step 3 to be still performed is at most* $\min\{y(H_0) - y(F_0), x(F_0) - x(H_0)\}$.

Let G^* be an optimal solution of a MCRT problem and G be a solution found by the algorithm. Then *the performance ratio of the algorithm w.r.t* G is $r(G) = y(G)/y(G^*)$.

Proposition 3.5 *Let the algorithm* $\mathcal{A}$ *find exact solutions. If our algorithm has terminated after a performance of Step 4 and* G *is the found solution, then* $r(G) \leq \bar{r}(G)$, *where*

$$\bar{r}(G) = y(G)/\left(y(G) + \frac{(x(G) - h)(y(G) - y(F))}{x(F) - x(G)}\right).$$

Proof: Let the line GF intersect the line $x = h$ in the point B and let A have coordinates $(h, y(G))$. Then the triangle GBA contains all optimal solutions. Indeed, no feasible solution can be to the right of the line AB, no optimal solution can be above the line AG, and no solution can belong to a line parallel to GF and "below" GF (any such solution G' would have $ax(G') + by(G') < c$).

Hence, for an optimal solution G^*, $y(B) \leq y(G^*)$. Now it is easy to check that $r(G) \leq y(G)/y(B) = \bar{r}(G)$. □.

4 Computational results

We have tested the behavior of our algorithm for the minimum cost restricted time assignment problem: given two $n \times n$-matrices $C = [c_{ij}]$ and $T = [t_{ij}]$ with non-negative entries and a non-negative real h, find a permutation $\pi \in S_n$ so that $\sum_{i=1}^{n} c_{i\pi(i)} = \min\{\sum_{i=1}^{n} c_{i\sigma(i)} : \sigma \in S_n,\ \sum_{i=1}^{n} t_{i\sigma(i)} \leq h\}$, where S_n is the set of all permutation on $\{1, 2, ..., n\}$. We randomly generated 60 instances of the minimum cost constrained time assignment problem with $n = 6, 8$ and 10. The entries of matrices varied from 0 to 99. The value of h was calculated by the formula $h = \max\{30n, n\mu/2\}$, where μ is a random integer from the interval [0,99]. For only five of the instances, Step 3 was not performed at all: two instances with $n = 6$ did not have any solution, an instance with $n = 8$ and two instances with $n = 10$ only needed Step 1 to find an optimal solution. Some results obtained for the rest 55 problems are reflected in Table 1: we show the number of instances with specified real performance ratio of the found solution (i.e. $r(G)$) and specified quality of the bound $\bar{r}(G)$ (i.e. $q(G) = \bar{r}(G)/r(G)$). Note that (for the 55 instances) the number of iterations, i, of Step 3 varied from 2 to 5 (moreover, $i = 5$ only for two instances).

Table 1: Number of instances with specified r and q.

n	$r = 1$	$1 < r \leq 1.1$	$r > 1.1$	$q \leq 1.1$	$1.1 < q \leq 1.2$	$q > 1.2$
6	14	3	1	11	7	0
8	8	8	3	9	7	3
10	7	7	4	16	2	0

We have also tested the behavior of our algorithm and Gvozdev's algorithm [3] for the two parameters minimum spanning tree (in complete graph) problem: given two symmetric $n \times n$-matrices $C = [c_{ij}]$ and $T = [t_{ij}]$ with non-negative entries and a non-negative real h, find a spanning tree $F = (V, E)$ in the complete graph K_n on vertices $\{1, 2, ..., n\}$ such that $\sum_{ij \in E} c_{ij} = \min\{\sum_{ij \in E'} c_{ij} : E' \in \mathbf{E}, \sum_{ij \in E'} t_{ij} \leq h\}$, where $\mathbf{E}$ is the family of all edge sets of spanning trees in K_n.

The matrices C and T were generated analogously to the corresponding matrices in the assignment problem above. They have orders n=20,30,40,50 and 60. The value of h was calculated by the formula $h = \max\{25n, 0.4n\mu\}$, where μ is a random integer from the interval [0,99].

Table 2 reflects, in a sense, the quality of solutions found by our algorithm. Not knowing optimal solutions in most of cases, we measure the quality by the bound $\bar{r}$ (see Proposition 3.5). One can see that the quality of the solutions steadily increases while the order n grows.

Table 2: Number of instances with specified $\bar{r}$.

n	$1 \leq \bar{r} \leq 1.02$	$1.02 < \bar{r} \leq 1.05$	$1.05 < \bar{r} \leq 1.1$	$\bar{r} > 1.1$
20	10	5	3	2
30	6	9	5	0
40	8	9	3	0
50	9	10	1	0
60	14	6	0	0
70	15	5	0	0

Table 3 shows some data which can be used to compare our algorithm with Gvozdev's algorithm [3]. In Table 3, m_1 is the number of instances when the costs of solutions found by our algorithm and Gvozdev's algorithm coincide, m_2 (m_3, respectively) is the number of instances when solutions obtained by our algorithm (Gvozdev's algorithm, respectively) are cheaper. As one can see, our algorithm produces, in many cases, cheaper solutions, especially for large values of n.

In Table 3, i_1 (i_3, respectively) denotes the average number of calls of a procedure to construct a minimum spanning tree in an ordinary weighted complete graph per instance required by our algorithm (Gvozdev's algorithm, respectively). The parameter i_2 is the maximum number of calls of the procedure above by our algorithm. These data show that the time required by our algorithm is usually about half of time required by Gvozdev's algorithm. In reality, Gvozdev's algorithm requires considerably more time than our algorithm as the computation of the parameters λ and $\Delta\lambda$ [3] in Gvozdev's algorithm (before starting iterations of the procedure) is quite time consumable.

Table 3: Comparison of our algorithm and Gvozdev's algorithm.

n	m_1	m_2	m_3	i_1	i_2	i_3
20	17	1	2	7.2	9	13.4
30	10	6	4	8.7	10	16.2
40	6	7	7	9.6	11	18.5
50	6	6	8	10.3	11	21.2
60	3	12	5	10.9	12	22.0
70	3	14	3	11.0	12	22.3

References

[1] Y.P. Aneja, V. Aggarwal and K.P.K. Nair, "Shortest chain subject to side constraints", *Networks* **13**, 295-302 (1983).

[2] M.R. Garey and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[3] S.E. Gvozdev, "On two discrete optimization problems", In *Upravliaemye systemy, Novosibirsk*, **19** 22-30 (1979) (in Russian).

[4] G. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem", *Networks* **10**, 293-310 (1980).

[5] R. Hassin, "Approximation schemes for the restricted shortest path problem", *Math. Oper. Res.* **17**, 36-42 (1992).

[6] M. Henig, "The shortest path problem with two objective functions", *European J. Oper. Res.* **25**, 281-291 (1985).

[7] H.C. Joksch, The shortest route problem with constraints, *J. Math. Anal. Appl.* **14**, 191-197 (1966).

[8] E.L. Lawler, *Combinatorial Optimization: Networks and Mathroids*, Hold, Rinehart and Winston, New York, 1976.

[9] I.I. Melamed, S.I. Sergeev, and I.Kh. Sigal, "The Traveling Salesman Problem: Part 1, Theoretical Issues", *Automation and Remote Control* **50** 1147-1173 (1989).

[10] B.M.E. Moret and H.D. Shapiro, "How to find a minimum spanning tree in practice", *Lecture Notes in CS* **555**, 192-203 (1991).

[11] S.A. Morozov, "On the minimum spanning tree problem with restrictions" In *Upravliaemye systemy, Novosibirsk*, **15**, 40-47 (1976) (in Russian).

[12] C.A. Phillips, "The network inhibition problem", in *Proc. 25th Ann. ACM Symp. Theory Computing*, 776-785 (1993).

[13] M.J. Rosenblatt, Z. Sinuani-Stern, "Generating the discrete efficient frontier to the capital budgeting problem", *Oper. Res.*, **37**, 384-394 (1987).

[14] H.A. Taha, *Operation Research: An Introduction*, New York, 1992.

[15] A. Warburton, "Approximation of Pareto optima in multi-objective, shortest path problems", *Oper. Res.* **35**, 70-79 (1987).