Egerváry Research Group
on Combinatorial Optimization

Technical reportS

TR-2001-17. Published by the Egrerváry Research Group, Pázmány P. sétány 1/C, H–1117, Budapest, Hungary. Web site: www.cs.elte.hu/egres. ISSN 1587–4451.

Optimization with additional variables and constraints

Alpár Jüttner

November 6, 2000

Optimization with additional variables and constraints ⋆

Alpár Jüttner ⋆⋆

Abstract

Norton, Plotkin and Tardos [12] proved that - loosely spoken, a linear programming problem can be solved in strongly polynomial time if, by deleting a constant number k of columns, it can be converted to a problem which is already known to be solvable in strongly polynomial time. In this paper, by a more careful application of Megiddo's technique, the running time of this algorithm is reduced from $O(Tq^{k+1})$ to $O(Tq^k)$, where T is the running time of the basic algorithm, and q is the number of the comparisons made by it. For small k it can cause significant improvement on the algorithms using method of [12]. This approach also improves the consequences of the result of [12].

1 Introduction

Solving combinatorial problems, one often comes across the problem when we are looking for a value $\lambda^* \in \mathbb{R}$ which is unknown for us, but we have a strongly polynomial separation algorithm $\mathcal{A}(\lambda)$ which is able to decide for a specific value of the parameter λ, whether $\lambda < \lambda^*$, $\lambda = \lambda^*$ or $\lambda > \lambda^*$ for example by answering -1, 0 or $+1$, and we want to compute the explicit value of the λ^*.

Without other restrictions, only approximation-like algorithms can be given for this problem. (For example it is easy to see that for $\lambda^* := \sqrt{2}$ there exists a separation algorithm that uses only comparisons, additions and multiplications of rational numbers and the input number, but λ^* cannot be obtained through these operations.)

Megiddo[9] showed, roughly if we avoid the multiplications in the separation algorithm, then λ^* can also be computed in strongly polynomial time. Namely, he proved the following theorem.

Theorem 1.1. *Suppose that we are given a $\lambda^* \in \mathbb{R}$ through a separation algorithm $\mathcal{A}(\lambda)$ which is able to decide whether $\lambda < \lambda^*$, $\lambda = \lambda^*$ or $\lambda > \lambda^*$ by answering -1, 0 or $+1$. If $\mathcal{A}$ is* linear in λ*, works in time T and takes at most q comparisons, then we can determine the value of λ^* in time $O(Tq)$.*

⋆Research supported by the Hungarian National Foundation for Scientific Research Grant, OTKA T029772.

⋆⋆Department of Operations Research, Eötvös University, Kecskeméti u. 10-12., Budapest, Hungary, H-1053 and Ericsson Traffic Laboratory, Laborc u.1, Budapest, Hungary H-1037. e-mail: alpar@cs.elte.hu

Note that the word *linear* does not refer to the running time in this paper but it means intuitively that the algorithm does not multiply to each other the elements of the input in which it is linear. The Section 2 is devoted to the precise definition of the notion of this *linearity condition*.

Although the previous theorem and the following ones fundamentally depend on the linearity assumption, it is not a strong restriction in the sense that most combinatorial optimization problems that can be solved in strongly polynomial time, can also be solved with a linear algorithm in strongly polynomial time.

Theorem 1.1 inspired a variety of applications, improvements, and extensions (see e.g. [11], [14], [3], [12], [1], [2]).

In higher dimension, a *separation algorithm* for a convex set $\mathcal{P}$ is a subroutine which decides if a given vector is in the set $\mathcal{P}$, and if not, gives a hyperplane that separates the given vector from $\mathcal{P}$. As an extension of Khachian's ellipsoid method [8], Grötchel, Lovász and Schrijver [4, 5], Karp and Papadimitriou [7] and Padberg and Rao [13] independently showed that a linear object function can be maximized in polynomial time over any polyhedron (having only rational vertices with known binary length) given by a separation algorithm. Unfortunately this method is not strongly polynomial.

Theorem 1.1 shows that in case when $\mathcal{P} \in \mathbb{R}$, if we have a strongly polynomial separation algorithm which is linear, then we can optimize in strongly polynomial time. Using similar idea, C. H. Norton, S. A. Plotkin and É. Tardos [12] extended this result to any higher (but fixed) dimension. Namely they proved the following theorem.

Theorem 1.2. *Let a closed convex set $\mathcal{P} \in \mathbb{R}^d$ be given through a separation algorithm which is linear in its input, runs in time T and makes at most q comparisons. Then there is an algorithm which in $O(Tq^d)$ time either finds a point $x \in \mathcal{P}$ maximizing cx, or concludes that $\{cx : x \in \mathcal{P}\}$ is unbounded from above.*

In [12] the authors also prove the following important consequence of Theorem 1.2.

Theorem 1.3. *Suppose that there exists an algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary b, which runs in time T, makes at most q comparisons and which is linear in b. Then for any fixed d, there is an algorithm which runs in $O(Tq^{d+1})$ time and solves the linear program $\max\{ax + cz : Ax + Cz \leq b\}$ for any matrix C with d columns.*

The idea of the proof is to use Theorem 1.2 by transforming the problem to a maximization problem over a convex set in $\mathbb{R}^{d+1}$ for which there exists a separation algorithm constructed from the basic algorithm.

Turning to the dual formulation of a certain problem we get that constant number of additional constraints can also be handled with this method, namely we get that

Theorem 1.4. *[12] Suppose that there exists an algorithm to solve the linear program $\max\{ax : Ax \leq b\}$ for arbitrary a, which runs in time T, makes at most q comparisons and which is linear in a. Then for any fixed d, there is an algorithm which runs in $O(Tq^{d+1})$ time and solves the linear program $\max\{ax : Ax \leq b, Cx \leq c\}$ for any vector c and matrix C with d rows.*

Note that there are no restrictions on the size of the linear program corresponding to the basic problem. Also, it doesn't have to be given explicitly, the only thing what we need is the existence of a linear algorithm which is able to solve the basic problem. Since most combinatorial optimization problem (such as matchings, b-factors or trees of a graph, flows, circulations, submodular flows, matroids, intersections of a pair of matroids etc) has a — maybe exponentially large — linear programming description, these theorems are quite useful tools for designing strongly polynomial algorithms, it addition to the theoretical importance that it widens the class of strongly polynomially solvable linear programs.

As an examples, let us see the following problem introduced by Shahrokhi and Matula [15].

Concurrent Multi-Commodity Flows. We are given a network $G = (V, E)$ with capacities c on its arcs, and a network of required demands $R = (V, F)$ with demands r on the arcs of R. A feasible solution to this problem is a collection of non-negative flows, f_{ij} for $(i, j) \in F$, all satisfying the same percentage of the corresponding demands. The objective is to maximize this percentage.

This problem can be easily formulated as a multi-commodity flow problem and a single additional column (see [12]). The multi-commodity problem can be solved in strongly polynomial time using the general result of É. Tardos[17]. Denoted the running time of this algorithm by T, since $q \leq T$, Theorem 1.3 provides an $O(T^3)$ time algorithm to the concurrent multi-commodity flow problem.

In a several other cases, the problem also has a small number of additional variables or constraints, typically one or two. In these cases any improvements in the exponent of the running time can be useful.

As the main result of this paper — by a more careful application of Megiddo's technique, an algorithm that reduces the running time from $O(Tq^{d+1})$ to $O(Tq^d)$ is presented in Section 3. The main idea of the proof is to apply Megiddo's technique directly instead of using Theorem 1.2, which makes possible to eliminate a redundant factor in the running time. Another advantage of this approach is to present a clear way to solve the nontrivial problem of obtaining the dual solution of the problem.

2 Linear Algorithms

To define the notation of the linear algorithm we use a RAM machine which has an additional storage called ***L**imited **A**ccess **M**emory*. It may store real numbers with a restriction that an algorithm running on this machine has only a limited access to this storage. Namely, it can reach the contents of the LAM only through the following operations.

- It can write an element of the RAM or LAM into an element of the LAM,
- it can multiply an element of the LAM with an element of the RAM,
- it can add an element of the LAM to another element of the LAM, and

- it can compare two elements of the LAM.

Note however, it cannot multiply two elements of the LAM and it cannot read them either (that is, it cannot copy an element of LAM into the RAM).

Definition 2.1. Let $\mathcal{A}(x, y)$ be an algorithm, where x and y are its input vectors. We say that $\mathcal{A}$ is **linear in** x if it gets x in the LAM, and we also expect the output in the LAM. The algorithm has full access to the other part of the input, in other words it gets it in the RAM.

It can be seen that the usual operations of the data structures can be implemented on a LAM machine, for example we can choose the minimal element of a set of numbers stored in the LAM and we can also sort its elements. However for example we cannot compute the determinant of a matrix with a LAM machine because we cannot avoid the multiplications of two elements of the matrix. (The determinant itself is a nonlinear polynomial of the elements of the matrix.)

3 Eliminating the additional variables

Consider the following slightly more general problem.

$$\max cx + dy \tag{1a}$$

where

$$Ax + By \leq b \tag{1b}$$
$$Ly \leq l \tag{1c}$$
$$Ey = e \tag{1d}$$

Let

$$\mathcal{R} := \{y \in \mathbb{R}^k : Ly \leq l, Ey = e\} \tag{2}$$

We prove, that

Theorem 3.1. *Suppose, that there exists an algorithm which solves the problem* $\max\{cx : Ax \leq b\}$*, is linear in* b*, runs in time* T *and makes at most* q *comparisons. Then for any fixed* k*, the problem (1a)-(1d) can be solved in time* $O(Tq^{\dim \mathcal{R}})$*.*

Proof. We prove the theorem by induction on $k := \dim \mathcal{R}$. Let suppose that the theorem holds whenever $\dim \mathcal{R} < k$.

First let $\mathcal{A}(A, B, b, c, y)$ be the algorithm which gets the matrices A and B and the vectors b, c and y, is linear in y and gives the optimal solution x^* to the problem

$$L(y) := \max cx + dy \tag{3a}$$

$$Ax \leq b - By \tag{3b}$$

for any *fixed* y, together with the dual optimal solution, which is an optimal solution z^* to the problem

$$L(y) = \min z(b - By) + dy \tag{4a}$$

$$z \geq 0 \tag{4b}$$

$$zA = c. \tag{4c}$$

Clearly, if we had an oracle which could give the second part y^* of an optimal solution to the problem (1a)-(1d), we could compute a (primal) optimal solution to the problem using $\mathcal{A}$. While an optimal place of y^* seems hard to compute directly, the linearity of $\mathcal{A}$ enables us to run it in such a way that at its each step it uses only an efficiently computable partial information on the optimal place of y^*. These partial information are computed using the following subroutine.

Claim 3.2 (Comparing subroutine). *Let us given a vector $v \in \mathbb{R}^k$ and a real number α. Then, using the induction hypothesis, we can choose a true one from the following three statements together with a certificate of its veracity.*

"$\leq$": $vy^* \leq \alpha$ holds for any optimal solution (x^*, y^*) to (1a)-(1d),

"$\geq$": $vy^* \geq \alpha$ holds for any optimal solution (x^*, y^*) to (1a)-(1d),

"$=$": there is an optimal solution (x^*, y^*) to (1a)-(1d) for which $vy^* = \alpha$.

Proof. First, using e.g. Megiddo's linear time linear programming method[1] [10] we check whether either $vy \leq \alpha$ or $vy \geq \alpha$ holds for all $y \in \mathcal{R}$ by maximizing/minimizing the function vy over the set $\mathcal{R}$. If one of these holds we return with "$\leq$" or "$\geq$" respectively and with the dual solution of the corresponding linear program as a certification. Then we check whether

$$vy = \alpha \tag{5}$$

holds for all $y \in \mathcal{R}$. If it holds, we return with "$=$" and also with the dual solution.

Otherwise, let $\mathcal{R}' := \{y \in \mathcal{R} : vy = \alpha\}$. Since $\dim \mathcal{R}' < \dim \mathcal{R}$, we are able to compute the primal and dual optimal solution to the system (1a)-(1d) and (5), so we have an optimal solution z^b, z^l, z^e, γ to the system

$$\min z^b b + z^l l + z^e e + \gamma\alpha \tag{6a}$$

$$z^b A = c \tag{6b}$$

$$z^b B + z^l L + z^e E + \gamma v = d. \tag{6c}$$

Then,

[1] The description of this algorithm in the book [16] also obtains the dual variables or the certificate of emptiness if the linear program is infeasible.

- If $\gamma = 0$ then z^b, z^l, z^e is also a feasible dual solution to the system (1a)-(1d) so the primal optimal solution is an optimal solution to (1a)-(1d), too. Thus we answer "=" and the vectors z^b, z^l, z^e.
- If $\gamma < 0$ then if we increase α the optimal object value of (6a)-(6c) will strictly decreasing, implying that $vy^* \leq \alpha$ for all optimal solution x^*, y^* to (1a)-(1d). So we answer "$\leq$" and give z^b, z^l, z^e, γ as the certificate of this fact.
- Similarly, if $\gamma > 0$ then we declare "$\geq$" and also give z^b, z^l, z^e, γ.

□

Let

$$\mathcal{Y} := \{y^* : \exists x^* \text{ such that } (x^*, y^*) \text{ is an optimal solution to (1a)-(1d)}\}. \tag{7}$$

We make a new algorithm $\mathcal{A}'$, which is a modification of $\mathcal{A}$. The algorithm sometimes calls the previous comparing algorithm. Whenever this subroutine is called with a vector v and a number α it either finds an optimal solution to (1a)-(1d) together with the dual solution (so we can stop) or states that whether $vy^* \leq \alpha$ or $vy^* \leq \alpha$ holds for each optimal $y^* \in \mathcal{R}$. We stores these resulted half-spaces. At a certain step of the algorithm let

$$\mathcal{Q} := \{y \in \mathcal{R} : v_i y \leq \alpha_i \ \forall i = 1, \cdots, t\}, \tag{8}$$

where v_i and α_i defines the half planes resulted by the comparing subroutine. Clearly, $\mathcal{Y} \subseteq \mathcal{Q}$.

First, we replace each element of the LAM with a pair of a real number and a k dimensional vector called *parametric numbers*. What we have in mind is that the parametric number (β, v) means the linear expression $\beta + vy^*$. Addition and subtraction of these numbers are defined in the same way as the usual vector addition and vector subtraction while the multiplications and divisions of these numbers are avoided by the linearity assumption on the algorithm $\mathcal{A}$.

The only undefined part of $\mathcal{A}'$ when it makes comparison, namely it inquires about whether $(\beta_1, v_1) \leq (\beta_2, v_2)$ or not. It means whether $\beta_1 + v_1 y^* \leq \beta_2 + v_2 y^*$ or not, which is equivalent to $(v_1 - v_2)y^* \leq \beta_2 - \beta_1$. Our aim is to give an answer which holds all $y^* \in \mathcal{Y}$. For this at first we test with Megiddo's linear-time algorithm whether either this or its negation is a consequence of the previous answers. If it is, we make decision according to this. If it isn't then we call the comparing subroutine with $v := (v_1 - v_2)$ and $\alpha := \beta_2 - \beta_1$. If the result is

"=" , then the comparing subroutine found a primal and a dual optimal solution to the system (1a)-(1d) so we can stop and return these data.

"$\leq$" , then we consider (β_1, v_1) *is less than or equal to* (β_2, v_2).

"$\geq$" , then we consider (β_1, v_1) *is greater than* (β_2, v_2).

Note that in the third case the decision is right only if there is no $y^* \in \mathcal{Y}$ for which $(v_1 - v_2)y^* = \beta_2 - \beta_1$.

Now, let us run $\mathcal{A}'(A, B, b, c, y')$, where y' is the vector of parametric numbers

$$y'_i := (0, (0, \cdots, 0, \overset{i^{th}}{1}, 0 \cdots, 0)). \tag{9}$$

Let

$$\mathcal{Q}' := \{y \in \mathcal{R} : y \text{ satisfies all previous decisions}\}, \tag{10}$$

that is, the set of $y \in \mathcal{R}$ for which $\mathcal{A}'$ certainly does the same as $\mathcal{A}'(A, B, b, c, y)$. Our comparing method ensures that $\mathcal{Q}' \neq \emptyset$ (that is the decisions are consistent), and $\overline{\mathcal{Q}'} = \mathcal{Q}$.

At the end of its running $\mathcal{A}'$ results a primal and a dual optimal solution x^y, z^y to (3a)-(3b) as *a linear function of* y, i.e. it results vectors x_0, z_0 and matrices X, Z, for which $x^y := x_0 + Xy$ and $z^y := z_0 + yZ$. These solutions must be right for all $y \in \mathcal{Q}'$, so they are also right for all $y \in \overline{\mathcal{Q}'} = \mathcal{Q}$, thus $L(y) = cx^y + dy$ for all $y \in \mathcal{Q}$.

Obviously $cx^y + dy$ is a linear function in y, namely $cx^y + dy = cx_0 + (cX + d)y$. The vector z^y is a continuous function of y and for each $y \in \mathcal{Q}$ the vector $d - z^y B$ is a $\sqrt{?}$supergradient of the concave function L in the point y, so it follows that

Claim 3.3. *For all $y' \in \mathcal{Q}$, the functions $f'_y(y) := (d - z^{y'}B)y + z^{y'}b$ are equal to $L(y)$ for all $y \in \mathcal{Q}$.* □

On the other hand $\mathcal{Y} \subseteq \mathcal{Q}$, so a vector y^* maximizing the function $L(y)$ (i.e. a $y^* \in \mathcal{Y}$) can be got by maximizing the function $(cX + d)y$ over the set $\mathcal{Q} = \{y : Ly \leq l, Ey = e, Vy \leq a\}$, where the rows of V and a correspond the calls of the comparing subroutine. It also can be done with Megiddo's linear time algorithm. Now, we call the original $\mathcal{A}$ with the parameters A, B, b, c, y^* let x^* and z^* be the resulted primal and dual solution. Then x^*, y^* is an *optimal primal solution* to the problem (1a)-(1d).

The only thing that we still have to do is to compute the *dual optimal solution* to the problem (1a)-(1d). Namely, we need vectors z_b^*, z_l^*, z_e^* for which

$$z_b^*, z_l^* \geq 0 \tag{11a}$$
$$z_b^* A = c \tag{11b}$$
$$z_b^* B + z_l^* L + z_e^* E = d \tag{11c}$$
$$z_b^* b + z_l^* l + z_e^* e \leq cx^* + dy^* \tag{11d}$$

Let $w := d - z^{y^*}B$. According to Claim 3.3 y^* is an optimal primal solution to the following problem.

$$\max wy \tag{12a}$$

$$Vy \leq a \tag{12b}$$
$$Ly \leq l \tag{12c}$$
$$Ey = e. \tag{12d}$$

Let z_b^M, z_l^M, z_e^M be the optimal dual solution to the previous problem, that is the optimal solution to the problem

$$\min z_1^M a + z_2^M l, z_3^M e \tag{13a}$$

$$z_1^M, z_2^M \geq 0 \tag{13b}$$

$$z_1^M V + z_2^M L + z_3^M E = d - z^{y^*} B. \tag{13c}$$

Now, let us define the following vectors

$$z_b^* := \frac{1}{\delta}\left(z^{y^*} - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^b\right) \tag{14a}$$

$$z_l^* := \frac{1}{\delta}\left(z_2^M - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^l\right) \tag{14b}$$

$$z_e^* := \frac{1}{\delta}\left(z_3^M - \sum_i \frac{z_1^M(i)}{\gamma_i} z_i^e\right), \tag{14c}$$

where z_i^b, z_i^l, z_i^e are the solutions to (6a)-(6c) corresponding to the rows of the matrix V and $\delta := 1 - \sum_i \frac{z_1^M(i)}{\gamma_i}$. Note, that $\gamma_i < 0$ for each i.

One can check, that (14a)-(14c) is a feasible solution to (11a)-(11c) and fulfills the complementary slackness condition, thus (11d) also holds.

Finally, the running time of the algorithm is the sum of the time used by $\mathcal{A}'$ itself and the time required by the comparisons made by it. Each comparison involves the same optimization task, but over a set $\mathcal{R}'$, the dimension of which is smaller by at least one than of $\mathcal{R}$. So, by the induction hypothesis each comparison runs in time $O(Tq^{\dim \mathcal{R}-1})$ and makes at most $q^{\dim \mathcal{R}-1}$ comparison. Since $\mathcal{A}'$ makes at most q comparisons, the total running time is $O(T + qTq^{\dim \mathcal{R}-1})$ and the algorithm makes at most $q \cdot q^{\dim \mathcal{R}-1}$, which yield the required running time. □□

Acknowledgement

The author wishes to thank András Frank for several valuable comments.

References

[1] R. E. Burkard, U. Pferschy. *The inverse-parametric knapsack problem.* EJOR 83 (1995) 376-393.

[2] R. E. Burkard, K. Dlaska, Bettina Klinz. *The Quickest Flow Problem.* ZOR Methods and Models of Operations Research, 37 (1993)1-58.

[3] E. Cohen, N. Megiddo. *Strongly Polynomial-time and NC Algorithms for Detecting Cycles in Dynamic Graphs*. Journal of the Association for Computing Machinery 40 (1993) 791-832.

[4] M. Grötschel, L. Lovász, and A. Schrijver. *The Ellipsoid Method and its Consequences in Combinatorial Optimization*. Combinatorica, 1:169-197,1981.

[5] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

[6] A. Jüttner. *On Budgeted Optimization Problem*. Submitted to Math. Oper. Res.

[7] R. M. Karp, C. H. Papadimitriou. *On Linear Characterization of Combinatorial Problems*. SIAM Journal of Computing, 11:620-632, 1982.

[8] L. G. Khachian. *Polynomial Algorithms in Linear Programming.* Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki, 20:53-72, 1980.

[9] N. Megiddo. *Combinatorial Optimization with Rational Objective Functions*. Math. Oper. Res., 4:414-424, 1979.

[10] N. Megiddo. *Linear Programming in Linear Time When the Dimension is Fixed.* Journal of the ACM, 31:114-127, 1984.

[11] N. Megiddo. *Applying Parallel Computation Algorithms in the Design of Serial Algorithms.* Journal of the ACM, 30:852-865, 1983.

[12] Carolyn Haibt Norton, Serge A. Plotkin, Éva Tardos. *Using Separation Algorithms in Fixed Dimension.* Journal of Algorithms 13 (1992), no 1, pp.79-98.

[13] M. W. Padberg, M. R. Rao. *The Russian Method for Linear Programming III.: Bounded integer programming.* Technical Report Research Report 81-39, New York University, Graduate School of Business Administration, 1981.

[14] T. Radzik. *Fractional combinatorial optimization*, In *Handbook of Combinatorial Optimization*, editors DingZhu Du and Panos Pardalos, vol. 1, Kluwer Academic Publishers, December 1998.

[15] F. Shahrokhi, D. W. Matula. *The Maximum Concurrent Flow Problem.* Technical Report CSR-183, Dept. of Computer Science, New Mexico Tech., 1988.

[16] A. Schrijver. *Theory of Linear and Integer Programming*, J. Wiley & Sons, 1986.

[17] Éva. Tardos. *A Strongly Polynomial Algorithm for Solving Combinatorial Linear Programs.* Operations Research, pages 250-256, 1986.