A POLYNOMIAL-TIME DESCENT METHOD FOR SEPARABLE CONVEX OPTIMIZATION PROBLEMS WITH LINEAR CONSTRAINTS

SERGEI CHUBANOV [*]

Abstract. We propose a polynomial algorithm for a separable convex optimization problem with linear constraints. We do not make any additional assumptions about the structure of the objective function except for polynomial computability. That is, the objective function can be non-differentiable. The running time of our algorithm is polynomial in the size of the input consisting of an instance of the problem and the accuracy with which the problem is to be solved. Our algorithm uses an oracle for solving auxiliary systems of linear inequalities. This oracle can be any polynomial algorithm for linear programming.

Key words. Convex optimization, separable convex function, polynomial algorithm

AMS subject classifications. 90C25

1. Introduction.

In this paper, we propose a polynomial algorithm for the problem of minimizing a separable convex function over a polyhedron. This is a well-known problem having applications in numerous areas (for some applications, see Hochbaum [10], Ahuja and Orlin [1], and Dantzig, Johnson, and White [8]).

The objective function, denote it by F, has the form $F(x) = \sum_j F_j(x_j)$, where F_j are convex functions of one variable. We assume that each function F_j is polynomially computable, which means that there is an algorithm for computing F_j whose running time is bounded by a polynomial in the size of the argument and in the size of an additional vector needed to compute the objective function. If F_j are piecewise quadratic convex or, more generally, piecewise polynomial, then this vector can contain the coefficients of the respective polynomials and a rule by which F_j should be computed. In general, functions F_j are given by an oracle. This means that even if F_j are piecewise linear, their explicit representation as a list of linear pieces may require a space which is exponential in the size of the input. For instance, if the problem contains a constraint $x_j \in [0, u_j]$, where $u_j \in \mathbb{Z}_+$, and F_j coincides with a strictly convex function at integer points and is linear between consecutive integer points, then a list of linear pieces must contain u_j items to describe F_j over $[0, u_j]$.

In traditional models of computation there are no finite algorithms that would be able to find exact optimal solutions to such problems because exact optimal solutions can have irrational components in the general case. So by a solution we understand a feasible solution minimizing the objective function with a given accuracy.

The running time of our algorithm is bounded by a polynomial in the binary size of the input consisting of the objective function, linear constraints, and the accuracy. To the best of our knowledge, except for the ellipsoid method, the algorithm presented in this paper is at the moment the only polynomial algorithm for the above problem.

As well as many other methods for convex optimization, such as for instance the Frank-Wolf method, also known as the conditional gradient method, our algorithm uses linear programming to find suitable descent directions. In the case of our algorithm, the respective auxiliary linear programs are homogeneous systems of linear inequalities. That is, at every iteration we only need to find a solution to the respective homogeneous system to proceed to the next iteration. If the problem in question

[*]University of Siegen, Germany, sergei.chubanov@uni-siegen.de

is a separable convex network flow problem, we in fact obtain a polynomial cycle-canceling algorithm because in this case, as we will see in Section 5, any algorithm for finding negative-cost cycles can be used to find descent directions.

Recently, it was observed that it is impossible to achieve a better dependence on ε than $O(\frac{1}{\varepsilon})$ in the running time when using a certain generic scheme based on the conditional gradient method; see Lan [14] and Garber and Hazan [12]. From this point of view, it is important to emphasize that there is an essential difference between the method presented in this paper and conditional gradient methods. While the conditional gradient methods move from one feasible solution to another according to a local linear model, our method uses a local *convex* model which can be written as a linear program in a higher dimensional space. At least in the separable case, this approach allows us to achieve a logarithmic dependence on $\frac{1}{\varepsilon}$ in the complexity estimate.

Our algorithm can use any polynomial algorithm for linear programming. One should note that the choice of polynomial algorithms for linear programming is not limited to the ellipsoid method and interior-point methods but also includes, for instance, the algorithms proposed by the author in [4, 5, 6].

If the objective function is differentiable, we can, of course, apply interior-point methods, under some additional assumptions. For example, the interior-point algorithm of Monteiro and Adler [18] runs in polynomial time if the objective function satisfies some conditions that are similar to self-concordance. Anyway, in the present paper we consider a simpler algorithm that is able to solve a much more general problem in polynomial time.

Some important special cases of our problem also include separable convex cost network flow problems. In this case there exist efficient algorithms whose running time is polynomial in the size of the input; see, for instance, Minoux [17], and Hochbaum and Shanthikumar [11]. Moreover, Tseng and Bertsekas [19] developed a polynomial algorithm for a separable convex generalized network flow problem. The algorithm of Hochbaum and Shanthikumar [11] is polynomial also in the case where the coefficient matrix is totally unimodular. Karzanov and McCormick [15] proposed a polynomial algorithm for a separable convex problem subject to constraints $Mx = \mathbf{0}$ with a totally unimodular coefficient matrix M.

Each iteration of the algorithm of Hochbaum and Shanthikumar [11] consists in approximating functions F_j by piecewise linear functions and solving the corresponding linear program. The number of variables in the auxiliary linear programs used by the algorithm of Hochbaum and Shanthikumar [11] depends linearly on the maximum absolute value of the subdeterminants of the coefficient matrix. This means that this algorithm is not polynomial for our problem. It is important to note that Hochbaum and Shanthikumar define the accuracy of a feasible solution x as $\|x - x^*\|_\infty$, where x^* is an optimal solution; a solution is called ρ-accurate if $\|x - x^*\|_\infty \le \rho$. We will show that our algorithm is able to find such ρ-accurate solutions as well, the running time remaining polynomial also in this case.

Dantzig [7] was probably the first to propose the idea of using piecewise linear approximations for separable convex functions and formulating the respective convex separable problems over polyhedra as linear programs. In general, when using this approach, we need an exponential number of linear functions, and, consequently, an exponential number of variables in the respective linear programs. For instance, even in the case of a separable quadratic objective function the minimum number of linear pieces needed to approximate it on a cube depends exponentially on the binary size of the required accuracy; see Güder and Morris [13].

Our algorithm is based on a more efficient method that consists in approximating the objective function only locally by replacing each F_j by a piecewise linear function composed of two linear functions. The advantage of this approach is that we consider linear programs whose number of variables is only twice as large as the number of variables of the original convex problem. The practical efficiency of the local approximation was demonstrated by Meyer [16]. However, he does not make any claims about the theoretical time complexity of his algorithm. Another example of application of the local piecewise linear approximation is the polynomial method for separable convex optimization in unimodular linear spaces of Karzanov and McCormick [15], which we have already mentioned. The aim of the present paper is to show that an appropriate choice of the local piecewise-linear approximations leads to a polynomial algorithm also in the general case.

2. Separable convex problem with linear constraints. Let $F(x) = \sum_{j=1}^{n} F_j(x_j)$ be a separable convex function, where functions $F_j : \mathbb{R} \to \mathbb{R}$ belong to some class $\mathcal{F}$ of convex functions defined on $\mathbb{R}$. (Note that all functions of this class are continuous.) For each function φ in $\mathcal{F}$, let $w(\varphi)$ be a rational vector, associated with φ, which is needed to compute φ. For instance, in the quadratic case this vector contains the coefficients of the respective polynomial. We assume that every function φ in $\mathcal{F}$ is polynomially computable. That is, we assume that there is an algorithm for computing $\varphi(\lambda)$ whose running time is polynomial in the size of $w(\varphi)$ and in the size of λ, provided that λ is rational. This means that there is a polynomial $p : \mathbb{R}^2 \to \mathbb{R}$ such that the number of arithmetic operations and the sizes of the intermediate numbers needed to compute φ are bounded by $p(size(\lambda), size(w(\varphi)))$ for every φ in $\mathcal{F}$, where *size* denotes the binary size. (Recall that the size of a rational number is the sum of the sizes of the numerator and denominator of the irreducible simple fraction representing this rational number.) Without loss of generality, we assume that each convex function of the form $\varphi = \max\{\varphi_1, \varphi_2\}$, where φ_1 and φ_2 are linear functions with rational coefficients, belongs to $\mathcal{F}$.

We consider the separable convex problem consisting of instances having the following form:

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

where A is a rational $m \times n$ matrix, b is a rational m-vector, and u is a rational n-vector. The $\mathbf{0}$ denotes a zero vector.

The set of instances of the above form, i.e., the separable convex problem, is completely determined by the respective class $\mathcal{F}$ of convex functions and the map w. So we can denote the separable convex problem by the pair $(\mathcal{F}, w)$.

Throughout the paper I denotes an instance of the separable convex problem $(\mathcal{F}, w)$. The binary size of I defined as

$$size(I) = size(A) + size(b) + size(u) + \sum_{j=1}^{n} size(w(F_j)).$$

The optimal value is denoted by $OPT(I)$. By an ε-approximate solution we understand a feasible solution x with $F(x) \le OPT(I) + \varepsilon$. In other words, ε is the required accuracy. The goal of our algorithm is to find an ε-approximate solution to I.

Further we will frequently use the well known fact that every convex function $\varphi : \mathbb{R} \to \mathbb{R}$ satisfies

$$\frac{\varphi(\gamma_2) - \varphi(\gamma_1)}{\gamma_2 - \gamma_1} \le \frac{\varphi(\gamma_4) - \varphi(\gamma_3)}{\gamma_4 - \gamma_3}, \quad \forall \gamma_1 < \gamma_2 \le \gamma_3 < \gamma_4. \tag{1}$$

If φ is differentiable, this inequality can be proved by applying the mean value theorem to the expressions at both sides of the inequality. In the general case, to prove the inequality, we can first consider the case $\gamma_2 = \gamma_3$ and apply Jensen's inequality. Then we use the case $\gamma_2 = \gamma_3$ to prove (1) in the general case.

The remarks below have no direct influence on the material of the other sections. They are only to demonstrate how to use our model in the case when the objective function is defined on an interval and when we have only an approximation oracle for computing the objective function with a given accuracy.

Indeed, the assumption that the convex functions F_j are defined on the whole real line is essential for our method. If we are going to use our method for a problem with a separable objective function $G(x) = \sum_{j=1}^{n} G_j(x_j)$ where G_j are convex functions defined over open or half-open intervals, then we need an appropriate reformulation so as to meet all the requirements on the objective function. For instance, let G_j be defined over $(0, u_j)$. Let θ be a sufficiently small number. Consider convex functions $H_j : \mathbb{R} \to \mathbb{R}$ defined as

$$H_j(x_j) = \begin{cases} G_j(x_j), & x_j \in [\theta, u_j - \theta], \\ \dfrac{G_j(\theta) - G_j(\theta/2)}{\theta/2}(x_j - \theta) + G_j(\theta), & x_j \in (-\infty, \theta), \\ \dfrac{G_j(u_j - \theta/2) - G_j(u_j - \theta)}{\theta/2}(x_j - u_j + \theta) + G_j(u_j - \theta), & x_j \in (u_j - \theta, \infty). \end{cases}$$

The convexity of H_j follows from the convexity of G_j. The function H_j is a convex extension of $G_j|_{[\theta, u_j - \theta]}$ to $\mathbb{R}$, where $G_j|_{[\theta, u_j-\theta]}$ is the restriction of G_j to $[\theta, u_j - \theta]$; the H_j is obtained from $G_j|_{[\theta, u_j-\theta]}$ by attaching two linear pieces. The functions H_j approximate G_j with the desired accuracy if θ is sufficiently small.

Let us demonstrate how we can choose θ in the case where each G_j is nonnegative and tends to infinity as x_j tends to 0 or to u_j. Let z with $\mathbf{0} < z < u$ be a feasible solution. Now, using a binary search, we can choose θ with $G_j(\theta) > G(z)$ and $G_j(u_j - \theta) > G(z)$ for all j. This ensures, in the mentioned case, that the set of optimal solutions of the problem with the objective function $H = \sum_j H_j$ coincides with the optimal set of the problem with the objective function G.

It is important to note that even if the objective function of a separable convex problem can only be computed with any given accuracy, the respective separable convex problem can be reformulated as a problem of the described class. To prove this statement, let us fix some value $\Delta > 0$ and consider a piecewise linear function φ which is linear in each interval $[s\Delta, (s+1)\Delta]$, $s \in \mathbb{Z}$, and such that, for some $\nu > 0$, $\psi : \mathbb{R} \to \mathbb{R}$, and $\tau : \mathbb{R} \to \mathbb{R}$,

$$\varphi(s\Delta) = \psi(s\Delta) + 2\nu(s\Delta)^2 + \tau(s\Delta), \quad \forall s \in \mathbb{Z}.$$

Lemma 1. *If ψ is convex and $|\tau(s\Delta)| \le \nu\Delta^2$ for all s, then φ is convex.*

Proof. It is easily verified that

$$s^2 - (s-1)^2 = (s+1)^2 - s^2 - 2$$

and

$$\tau(s\Delta) - \tau((s-1)\Delta) - 4\nu\Delta^2 \le -2\nu\Delta^2 \le \tau((s+1)\Delta) - \tau(s\Delta).$$

The convexity of ψ implies

$$\psi(s\Delta) - \psi((s-1)\Delta) \le \psi((s+1)\Delta) - \psi(s\Delta).$$

So we have

$$\begin{aligned}
&\varphi(s\Delta) - \varphi((s-1)\Delta) \\
&= \psi(s\Delta) - \psi((s-1)\Delta) + 2\nu\Delta^2(s^2 - (s-1)^2) + \tau(s\Delta) - \tau((s-1)\Delta) \\
&\le \psi((s+1)\Delta) - \psi(s\Delta) + 2\nu\Delta^2((s+1)^2 - s^2) + \tau(s\Delta) - \tau((s-1)\Delta) - 4\nu\Delta^2 \\
&\le \psi((s+1)\Delta) - \psi(s\Delta) + 2\nu\Delta^2((s+1)^2 - s^2) + \tau((s+1)\Delta) - \tau(s\Delta) \\
&= \varphi((s+1)\Delta) - \varphi(s\Delta).
\end{aligned}$$

It follows that

$$\varphi(s\Delta) - \varphi((s-1)\Delta) \le \varphi((s+1)\Delta) - \varphi(s\Delta), \quad \forall s \in \mathbb{Z}.$$

Therefore, taking into account that φ is piecewise linear, we conclude that φ is convex. ■

Let $\psi : \mathbb{R} \to \mathbb{R}$ be a convex function given by a vector $\omega(\psi)$ encoding the information about the function. Assume that there is an approximation oracle which is able to compute ψ with any given accuracy. More precisely, we assume that, for a given γ and an accuracy $\beta > 0$, the oracle returns a rational value ξ_β with $|\xi_\beta - \psi(\gamma)| \le \beta$. Moreover, we assume that in the case of rational values γ and β the algorithm runs in time that is polynomial in $size(\omega(\psi))$, $size(\gamma)$, and $size(\beta)$.

On the other hand, this approximation oracle is an (exact) oracle for computing the function $\psi^{\#}_{\beta} : \mathbb{R} \to \mathbb{R}$ such that, for each γ, $\psi^{\#}_{\beta}(\gamma)$ is equal to exactly the value ξ_β which would be returned by the approximation oracle if it was called to calculate $\psi(\gamma)$ with accuracy β. This function can be written as

$$\psi^{\#}_{\beta}(\gamma) = \psi(\gamma) + \tau_\beta(\gamma),$$

where τ_β is some function with $|\tau_\beta(\gamma)| \le \beta$ for all γ.

In general, $\psi^{\#}_{\beta}(\gamma)$ is not convex. However, on the basis of this function it is possible to construct a polynomially computable convex function approximating ψ with a given accuracy in a given interval. W.l.o.g. suppose we would like to approximate ψ over the interval $[0, 1]$.

The convexity of ψ implies that

$$\max\{|\psi(0) - \psi(-1)|, |\psi(3) - \psi(2)|\} \tag{2}$$

is an upper bound on the slopes of the supporting lines to the graph of ψ over $[0, 2]$. (Here we consider the larger interval $[0, 2]$ by technical reasons.) The following value is an upper bound on (2):

$$\begin{aligned}
K = \max\{&|\psi^{\#}_{1}(0) - \psi^{\#}_{1}(-1) + 2|, |\psi^{\#}_{1}(-1) - \psi^{\#}_{1}(0) + 2|, \\
&|\psi^{\#}_{1}(3) - \psi^{\#}_{1}(2) + 2|, |\psi^{\#}_{1}(2) - \psi^{\#}_{1}(3) + 2|\}.
\end{aligned}$$

The value K is polynomially computable.

Let ρ be a positive rational number and

$$\Delta := \frac{\min\{1,\rho\}}{4\max\{1,K\}},\ \nu := \frac{\rho}{22}.$$

Define φ as a piecewise linear function which is linear over the intervals $[s\Delta,(s+1)\Delta]$ and such that

$$\varphi(s\Delta) = \psi^{\#}_{\nu\Delta^2}(s\Delta) + 2\nu(s\Delta)^2, \quad \forall s \in \mathbb{Z}.$$

Define $\omega(\varphi) := (\omega(\psi),\rho)$.

THEOREM 2. *The function φ defined as above is convex, satisfies the condition*

$$|\psi(\gamma) - \varphi(\gamma)| \le \rho, \quad \forall \gamma \in [0,1],$$

and can be computed in time polynomial in the size of the argument and size$(\omega(\varphi))$.

Proof. The function φ is convex by Lemma 1 because ψ is convex and $|\tau_{\nu\Delta^2}(\gamma)| \le \nu\Delta^2$ for all γ. The polynomial computability of φ follows from the polynomial computability of $\psi^{\#}_{\nu\Delta^2}$.

Let $\gamma \in [0,1]$ and $s = \lfloor \frac{\gamma}{\Delta} \rfloor$. Since φ is linear over $[s\Delta,(s+1)\Delta]$ and we have $\Delta^2 \le \Delta \le 1$ and $s\Delta \in [0,1]$, we obtain

$$\begin{aligned}|\varphi(\gamma) - \varphi(s\Delta)| &\le |\varphi((s+1)\Delta) - \varphi(s\Delta)| \\ &= |\psi((s+1)\Delta) - \psi(s\Delta) + \tau_{\nu\Delta^2}((s+1)\Delta) - \tau_{\nu\Delta^2}(s\Delta) + 2\nu(2s+1)\Delta^2| \\ &\le |\psi((s+1)\Delta) - \psi(s\Delta)| + 2\nu\Delta^2 + 2\nu(2s+1)\Delta^2 \le K\Delta + 8\nu\end{aligned}$$

because

$$|\psi((s+1)\Delta) - \psi(s\Delta)| \le K\Delta$$

due to the fact that $[s\Delta,(s+1)\Delta]$ is a subset of $[0,2]$ and K is an upper bound on (2). By the same reason,

$$|\psi(s\Delta) - \psi(\gamma)| \le K(\gamma - s\Delta) \le K\Delta.$$

Then

$$|\varphi(s\Delta) - \psi(\gamma)| = |\psi(s\Delta) + \tau_{\nu\Delta^2}(s\Delta) + 2\nu(s\Delta)^2 - \psi(\gamma)| \le 3\nu + K\Delta.$$

It follows that

$$|\varphi(\gamma) - \psi(\gamma)| = |\varphi(\gamma) - \varphi(s\Delta) + \varphi(s\Delta) - \psi(\gamma)| \le 11\nu + 2K\Delta \le \rho.$$

■

It follows that any convex separable function which is computable by means of an approximation oracle can be approximated with a given accuracy in a given interval by a *convex* function for which there is an *exact* oracle. Therefore, the range of applications of our model also includes separable convex problems where the objective functions are given by polynomial-time approximation oracles.

3. Approximation by linear problems.

The basic idea of our algorithm is as follows. Consider a feasible solution x^0. Now, our goal is to find a new feasible solution x' such that $F(x^0) > F(x')$. To do this, we construct a convex function L

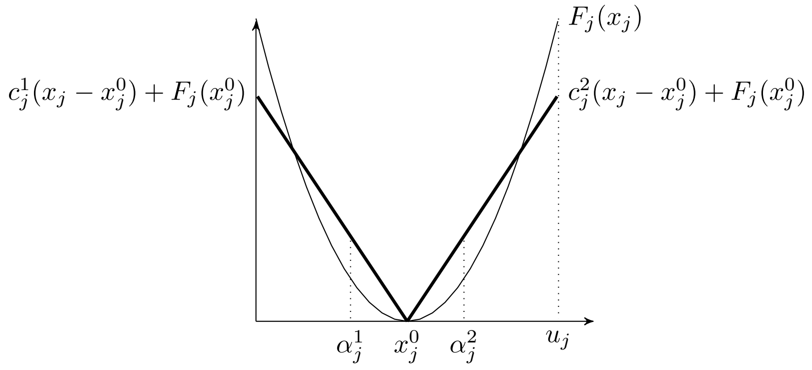

$F_j(x_j)$
$c_j^1(x_j - x_j^0) + F_j(x_j^0)$
$c_j^2(x_j - x_j^0) + F_j(x_j^0)$
α_j^1
x_j^0
α_j^2
u_j

FIGURE 1. *The local two-segment approximation for F_j around x_j^0. The thick broken line is the graph of L_j. Function L_j is the maximum of the linear functions $c_j^1(x_j - x_j^0) + F_j(x_j^0)$ and $c_j^2(x_j - x_j^0) + F_j(x_j^0)$.*

such that the epigraph of L has the form of the set $(x^0; F(x^0)) + C$ where C is a cone. Additionally, we require that

$$L(x) \leq F(x) + n\delta \tag{3}$$

for all feasible x and there exists a neighborhood $\mathcal{S}$ of x^0 such that

$$L(x) - F(x) \geq \mathit{const} \cdot \delta, \quad \forall x \in \partial\mathcal{S}, \tag{4}$$

where $\partial\mathcal{S}$ denotes the boundary of $\mathcal{S}$, δ is a positive value, and *const* denotes a positive constant. If we find a feasible descent direction of L at x^0, then we simply move from x^0 in this direction to a point x' in the boundary of $\mathcal{S}$. Since the structure of the epigraph of L implies that L is linear over each ray starting at x^0, we have $L(x^0) > L(x')$. Moreover, the structure of L implies that $L(x^0) = F(x^0)$. Therefore, the inequality (4) implies that

$$F(x') \leq L(x') - \mathit{const} \cdot \delta < L(x^0) - \mathit{const} \cdot \delta = F(x^0) - \mathit{const} \cdot \delta.$$

Then we set $x^0 := x'$ and perform the next iteration. This step improves the current value of F by at least $\mathit{const} \cdot \delta$. The δ is chosen so that the gap between the initial objective value and the optimal value is bounded by $2n\delta$. Therefore, we need only $O(n)$ improvements of the objective value until we can prove that x^0 is a minimum of L. In this case, for any optimal solution x^*,

$$F(x^0) = L(x^0) \leq L(x^*) \leq F(x^*) + n\delta,$$

which follows from (3). If $\delta \leq \frac{\varepsilon}{n}$, then the current solution is ε-approximate. Otherwise, we divide δ by two and continue the process until $\delta \leq \frac{\varepsilon}{n}$.

The following construction is a realization of the above general scheme in the case of our separable convex problem. Let x^0 be a point with $Ax^0 = b$ and $\mathbf{0} < x^0 < u$. (Using a polynomial algorithm for linear programming, we can always reformulate the problem so that such a solution exists.) Now we construct a separable convex function L that approximates F in some neighborhood of x^0. We define L as

$$L(x) = \sum_{j=1}^{n} L_j(x_j),$$

where

$$L_j(x_j) = \max\left\{c_j^1(x_j - x_j^0) + F_j(x_j^0),\ c_j^2(x_j - x_j^0) + F_j(x_j^0)\right\}. \tag{5}$$

Here, $c_j^1 < c_j^2$, $j = 1, \ldots, n$, and the coefficients c_j^1 and c_j^2 are chosen so that

$$\frac{\delta}{2} < \max\left\{c_j^1(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j) \mid x_j \in [0, x_j^0]\right\} \leq \delta, \tag{6}$$

and

$$\frac{\delta}{2} < \max\left\{c_j^2(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j) \mid x_j \in [x_j^0, u_j]\right\} \leq \delta. \tag{7}$$

The existence of such a function L will be proved later; see Lemma 4. Note that $L(x^0) = F(x^0)$. Note that the functions L_j are convex. Consider a value α_j^1 in $(0, x_j^0)$ such that

$$L_j(\alpha_j^1) - F_j(\alpha_j^1) \geq \frac{\delta}{2} \tag{8}$$

and α_j^2 in (x_j^0, u_j) such that

$$L_j(\alpha_j^2) - F_j(\alpha_j^2) \geq \frac{\delta}{2}. \tag{9}$$

The existence of α_j^1 and α_j^2 follows from (6) and (7). Figure 1 illustrates our construction.

Now we will show how to choose suitable coefficients c_j^1 and c_j^2 and the values α_j^1 and α_j^2 so as to guarantee that their sizes are bounded by a polynomial in *size*(I), *size*(δ), and *size*(x^0).

Further, in all complexity estimates, we will use the convention that $\log \gamma$ should read as $\max\{1, \log_2 \gamma\}$.

First, we need the following lemma (although this lemma is easy to prove by a standard binary-search argument, we give it here with a proof because we will need some details):

Lemma 3. *Let $f : \mathbb{R} \to \mathbb{R}$ be a concave function. Consider a^- and a^+ with $a^- < a^+$. For every $\rho > 0$, we can find λ in (a^-, a^+) such that*

$$f(\lambda) \geq \max_{[a^-, a^+]} f - \rho$$

in

$$O\left(\log \frac{(a^+ - a^-)K}{\rho}\right)$$

arithmetic operations and computations of f, where K is an upper bound on the absolute value of the slopes of the supporting lines to the graph of f over $[a^-, a^+]$.

Proof. Let $a_1 = a^-$ and $a_2 = a^+$. Our procedure will reduce the length of $[a_1, a_2]$ with each iteration, preserving the property that $[a_1, a_2]$ contains a maximum of f over $[a^-, a^+]$. Consider a' and a'' in $[a_1, a_2]$ with $a' < a''$ that divide $[a_1, a_2]$ into three equal parts. That is, $a' - a_1 = a'' - a' = a_2 - a''$. If $f(a') < f(a'')$, then $[a', a_2]$ contains a point delivering the maximum of f, because f is concave. Otherwise, $[a_1, a'']$ contains such a point. Depending on which of the two cases occurs, replace the current interval $[a_1, a_2]$ either by $[a_1, a'']$ or by $[a', a_2]$. This reduces its length by a factor of 3/2. We perform iterations of the described procedure until $a_2 - a_1 \le \rho/K$. Then we choose

$$\lambda = (a_1 + a_2)/2.$$

This value belongs to the interior of the original interval $[a^-, a^+]$ and has the property that $f(\lambda) \ge \max_{[a^-,a^+]} f - \rho$ because the current interval $[a_1, a_2]$ contains a maximum point of f and the value of f at this point differs by at most ρ from each value of f over $[a_1, a_2]$ due to the stopping condition of the above procedure. At the last iteration,

$$a_1 = a^- + \sum_{t=1}^{r} \frac{p_t}{3} \cdot \frac{2^{t-1}}{3^{t-1}}(a^+ - a^-),$$

where r is the number of iterations and $p_t \in \{0, 1\}$ for each t. The coefficient p_t indicates whether a_1 remains the same at the respective iteration ($p_t = 0$) or is increased by one third of the length of the current interval ($p_t = 1$). Since each iteration reduces the length of the current interval by a factor of 3/2, at the last iteration we have

$$a_2 - a_1 = \frac{2^r}{3^r}(a^+ - a^-).$$

Therefore, the λ found at the last iteration has the form

$$\lambda = a_1 + \frac{a_2 - a_1}{2} = a^- + \sum_{t=1}^{r} \frac{2^{t-1}p_t}{3^t}(a^+ - a^-) + \frac{2^{r-1}}{3^r}(a^+ - a^-). \tag{10}$$

The number of iterations r is bounded by $O\left(\log \frac{(a^+ - a^-)K}{\rho}\right)$, which implies the required estimate of the running time. ■

The following procedure is the one described in the proof of the above lemma:

```
**Procedure for maximizing a concave function $f$ with accuracy $\rho$**
$[a_1, a_2] := [a^-, a^+]$;
$a' := a^- + (a^+ - a^-)/3$;
$a'' := a^- + 2(a^+ - a^-)/3$;
repeat
    if $f(a') < f(a'')$ then $a_1 := a'$; else $a_2 := a''$;
    $a' := a_1 + (a_2 - a_1)/3$;
    $a'' := a_1 + 2(a_2 - a_1)/3$;
until $a_2 - a_1 \le \rho/K$
Return $\lambda = (a_1 + a_2)/2$.
```

Using inequality (1), it is not hard to prove that the following value is an upper bound on the absolute values of the slopes of the supporting lines to the graphs of the functions F_j over the respective intervals $[0, u_j]$:

$$K_{\max} = \max_j \{|F_j(-1) - F_j(0)|, |F_j(u_j + 1) - F_j(u_j)|\}. \tag{11}$$

Note that $size(K_{\max})$ is polynomially bounded in $size(I)$ because F_j are polynomially computable. The value $K_{\max}$ has the property that

$$|F_j(x''_j) - F_j(x'_j)| \le K_{\max}(x''_j - x'_j) \tag{12}$$

for any x''_j and x'_j in $[0, u_j]$ such that $x''_j > x'_j$. We will use this property to prove the correctness of the following two procedures that compute c^1_j and c^2_j; see the proof of Lemma 4:

```
**Procedure for $c^1_j$ and $\alpha^1_j$**
$[c'_j, c''_j] := [-K_{\max} - \frac{2\delta}{x^0_j}, K_{\max}]$;
**do**
    $c_j := (c'_j + c''_j)/2$;
    $f(x_j) := c_j(x_j - x^0_j) + F_j(x^0_j) - F_j(x_j)$;
    $[a^-, a^+] := [0, x^0_j]$;
    $\rho := \frac{\delta}{8}$;
    Maximize $f$ with accuracy $\rho$;
    Let $\lambda$ be the value returned by the respective procedure;
    **if** $\frac{5\delta}{8} \le f(\lambda) \le \frac{7\delta}{8}$ **then** return $c^1_j = c_j$ and $\alpha^1_j = \lambda$;
    **if** $f(\lambda) > \frac{7\delta}{8}$ **then** $c'_j := c_j$; **else** $c''_j := c_j$;
**until** $c''_j - c'_j \le \frac{\delta}{8x^0_j}$
Return $c^1_j = c_j$ and $\alpha^1_j = \lambda$.
```

```
**Procedure for $c^2_j$ and $\alpha^2_j$**
$[c'_j, c''_j] := \left[-K_{\max}, K_{\max} + \frac{2\delta}{(u_j - x^0_j)}\right]$;
**do**
    $c_j := (c'_j + c''_j)/2$;
    $f(x_j) := c_j(x_j - x^0_j) + F_j(x^0_j) - F_j(x_j)$;
    $[a^-, a^+] := [x^0_j, u_j]$;
    $\rho := \frac{\delta}{8}$;
    Maximize $f$ with accuracy $\rho$;
    Let $\lambda$ be the value returned by the respective procedure;
    **if** $\frac{5\delta}{8} \le f(\lambda) \le \frac{7\delta}{8}$ **then** return $c^2_j = c_j$ and $\alpha^2_j = \lambda$;
    **if** $f(\lambda) > \frac{7\delta}{8}$ **then** $c''_j := c_j$; **else** $c'_j := c_j$;
**until** $c''_j - c'_j \le \frac{\delta}{8(u_j - x^0_j)}$
Return $c^2_j = c_j$ and $\alpha^2_j = \lambda$.
```

LEMMA 4. *The above procedures find coefficients c^1_j and c^2_j with (6) and (7) and the values α^1_j and α^2_j with (8) and (9) in*

$$O\left(n\left(\log \frac{K_{\max}\|u\|_\infty}{\delta}\right)^2\right)$$

arithmetic operations and computations of functions F_j. The sizes of all the numbers considered in the course of the above procedures are polynomially bounded in size$(K_{\max})$, size(δ), size(u), and size(x^0).

Proof. It is sufficient to prove the correctness of the procedure for finding c^2_j and α^2_j. (To find c^1_j and α^1_j, we can change the coordinate system, use the procedure for c^2_j and α^2_j, and then express the result in the original coordinate system.)

Our procedure for finding a coefficient c_j^2 with (7) is based on a binary search in some interval $[c_j^-, c_j^+]$ which contains all c_j^2 with (7). First we will show that the coefficients of the linear functions g^- and g^+ defined below can be chosen as the endpoints of that search interval. Then we will describe the binary search procedure.

Determining the search interval. Consider linear functions

$$g^-(x_j) := -K_{\max}(x_j - x_j^0) + F_j(x_j^0)$$

and

$$g^+(x_j) := \left(K_{\max} + \frac{2\delta}{u_j - x_j^0}\right)(x_j - x_j^0) + F_j(x_j^0).$$

The inequality (12) implies that

$$(13) \qquad -K_{\max} \le \frac{F_j(x_j) - F_j(x_j^0)}{x_j - x_j^0} \le K_{\max}, \quad \forall x_j \in (x_j^0, u_j].$$

Hence,

$$g^-(x_j) \le F_j(x_j) \le g^+(x_j), \quad \forall x_j \in [x_j^0, u_j].$$

This implies that

$$(14) \qquad \max\left\{g^-(x_j) - F_j(x_j) \mid x_j \in [x_j^0, u_j]\right\} \le 0.$$

Moreover, taking into account (13) for $x_j = u_j$, we can write

$$(15) \qquad \max\left\{g^+(x_j) - F_j(x_j) \mid x_j \in [x_j^0, u_j]\right\} \ge g^+(u_j) - F_j(u_j) \ge 2\delta > \delta.$$

Let c_j^2 satisfy (7). Consider the linear function

$$g(x_j) = c_j^2(x_j - x_j^0) + F_j(x_j^0).$$

The inequality (7) implies

$$g(u_j) \le F_j(u_j) + \delta, \quad \exists x_j' \in [x_j^0, u_j] : g(x_j') \ge F_j(x_j') + \frac{\delta}{2}.$$

On the other hand, the inequalities (15) and (14) imply

$$g^+(u_j) > F_j(u_j) + \delta, \quad g^-(x_j') \le F_j(x_j').$$

It follows that

$$g(u_j) < g^+(u_j), \quad g(x_j') > g^-(x_j').$$

It follows that function g is not less than g^- and not greater than g^+ on $[x_j^0, u_j]$ (because for any two linear functions g_1 and g_2 with $g_1(x_j^0) = g_2(x_j^0)$ there are only two variants: $g_1(x_j) \ge g_2(x_j)$ for all $x_j \in (x_j^0, u_j]$ *or* $g_1(x_j) \le g_2(x_j)$ for all $x_j \in (x_j^0, u_j]$). Thus, for every c_j^2 which satisfies (7) we have

$$g^-(x_j) \le c_j^2(x_j - x_j^0) + F_j(x_j^0) \le g^+(x_j), \quad \forall x_j \in [x_j^0, u_j].$$

Therefore, any c_j^2 satisfying (7) belongs to the interval

$$[c_j^-, c_j^+] := \left[-K_{\max}, K_{\max} + \frac{2\delta}{u_j - x_j^0}\right].$$

The binary search. Consider $\varphi : \mathbb{R} \to \mathbb{R}$ defined as

$$\varphi(c_j) = \max\left\{c_j(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j) \mid x_j \in [x_j^0, u_j]\right\}$$

for all c_j. The function φ is continuous and nondecreasing. Since (14) and (15) take place, we have $\varphi(c_j^-) \leq 0$ and $\varphi(c_j^+) > \delta$. Then the interval $[c_j^-, c_j^+]$ contains c_j^* with

$$\varphi(c_j^*) = \frac{7\delta}{8}.$$

That is, $c_j^2 = c_j^*$ satisfies (7). Now our goal is to find a coefficient which would be sufficiently close to c_j^*. For this we perform a binary search in the interval $[c_j^-, c_j^+]$. Further in the proof, $[c_j', c_j'']$ denotes a subinterval containing c_j^*. Its length will be reduced by a factor of 2 with each iteration of the binary-search procedure described below. So let us start with

$$[c_j', c_j''] := [c_j^-, c_j^+].$$

Let

$$c_j = \frac{c_j' + c_j''}{2}.$$

Apply Lemma 3 with

$$\rho = \frac{\delta}{8},\ K = K_{\max} + |c_j|,\ [a^-, a^+] = [x_j^0, u_j], \tag{16}$$

and the function f defined as

$$f(x_j) = c_j(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j).$$

We can write $\varphi(c_j)$ as

$$\varphi(c_j) = \max_{[x_j^0, u_j]} f.$$

Now we apply Lemma 3. Let λ be the value found by the procedure described in the proof of Lemma 3. Consider the following three cases.

Case 1. $\frac{5\delta}{8} \leq f(\lambda) \leq \frac{7\delta}{8}$:

$$\frac{\delta}{2} < f(\lambda) \leq \varphi(c_j) \leq f(\lambda) + \rho \leq \delta.$$

In this case $c_j^2 = c_j$ satisfies (7) and we are done.

Case 2. $f(\lambda) < \frac{5\delta}{8}$:

$$\varphi(c_j) \leq f(\lambda) + \rho \leq \frac{5\delta}{8} + \frac{\delta}{8} = \frac{3\delta}{4} < \frac{7\delta}{8} = \varphi(c_j^*).$$

In this case $c_j^* \in [c_j, c_j'']$ because φ is nondecreasing. We set $c_j' := c_j$. The new interval $[c_j', c_j'']$ contains c_j^*.

Case 3. $f(\lambda) > \frac{7\delta}{8}$:

$$\varphi(c_j) \geq f(\lambda) > \frac{7\delta}{8} = \varphi(c_j^*).$$

In this case $c_j^* \in [c_j', c_j]$ because φ is nondecreasing. We set $c_j'' := c_j$. The new interval $[c_j', c_j'']$ contains c_j^*.

We continue the iterations until Case 1 occurs (and then we simply choose $c_j^2 = c_j$) or

$$c_j'' - c_j' \leq \frac{\delta}{8 \cdot (u_j - x_j^0)}. \tag{17}$$

It remains to consider the latter case. By the definition of φ, there is x_j in $[x_j^0, u_j]$ such that

$$\varphi(c_j'') = c_j''(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j).$$

For the same x_j,

$$\varphi(c_j') \geq c_j'(x_j - x_j^0) + F_j(x_j^0) - F_j(x_j).$$

It follows that

$$\varphi(c_j'') - \varphi(c_j') \leq (c_j'' - c_j')(x_j - x_j^0) \leq (c_j'' - c_j')(u_j - x_j^0) \leq \frac{\delta}{8}.$$

Then, since $[c_j', c_j'']$ contains c_j^* and φ is nondecreasing,

$$\varphi(c_j') \leq \varphi(c_j^*) \leq \varphi(c_j'') \leq \varphi(c_j') + \frac{\delta}{8}. \tag{18}$$

Choose $c_j^2 = \frac{c_j'' + c_j'}{2}$. In this case, $c_j^2 \in [c_j', c_j'']$. Then the inequalities (18) hold true also for this c_j^2 in place of c_j^*. Hence

$$|\varphi(c_j^*) - \varphi(c_j^2)| \leq \frac{\delta}{8}$$

for the chosen coefficient c_j^2. Then, recalling $\varphi(c_j^*) = \frac{7\delta}{8}$, we obtain

$$\frac{3\delta}{4} \leq \varphi(c_j^2) \leq \delta.$$

In other words, such a coefficient c_j^2 satisfies (7). We also set $\alpha_j^2 := \lambda$ where λ is the value λ found at the last iteration of the above procedure. Lemma 3 implies that $\lambda \in (x_j^0, u_j)$. We have found the required c_j^2 and α_j^2.

The complexity of the procedure. Note that the length of $[c_j^-, c_j^+]$ is equal to $2K_{\max} + \frac{2\delta}{u_j - x_j^0}$. The described procedure terminates no later than the length of the interval $[c_j', c_j'']$ becomes less than $\frac{\delta}{8 \cdot (u_j - x_j^0)}$, due to (17), which implies that the number of iterations is bounded by $O\left(\log \frac{K_{\max}(u_j - x_j^0)}{\delta}\right)$. That is, since $u_j - x_j^0 \leq \|u\|_\infty$, the procedure requires $O\left(\log \frac{K_{\max}\|u\|_\infty}{\delta}\right)$ iterations. Lemma 3 and the choice of parameters in (16) imply the same estimate for the number of iterations of the procedure for the approximate maximization of concave functions whenever it is called from the described procedure.

At the end of iteration k of the described procedure,

$$c_j' = c_j^- + \sum_{t=1}^{k} \frac{p_k}{2^k}(c_j^+ - c_j^-),$$

where $p_k \in \{0, 1\}$, and

$$c_j'' = c_j' + \frac{c_j^+ - c_j^-}{2^k}.$$

Since k is bounded by a polynomial in $size(K_{\max})$, $size(u)$, and $size(\delta)$, it follows that the sizes of both c'_j and c''_j, and, consequently, of c_j, are polynomial in $size(K_{\max})$, $size(\delta)$, $size(u)$, and $size(x^0)$, in the course of the procedure. The equation (10) implies that the size of the value α_j^2 returned by the procedure is bounded by a polynomial in $size(K_{\max})$, $size(\delta)$, $size(u)$, and $size(x^0)$ because r in (10) is polynomially bounded in these values whenever the procedure for the approximate maximization of concave functions is called from the procedure for finding c_j^2 and α_j^2. The sizes of the values a_1, a_2, a', and a'' in the course of that maximization procedure are polynomially bounded in the same sense. The same can be said about the sizes of $f(a')$ and $f(a'')$, due to the polynomial bound on $size(c_j)$ and the polynomial computability of F_j.

The c_j^1 and α_j^1 can be calculated in a similar way. The respective procedure is given just before this lemma. So we obtain the required bound on the running time needed for the calculation of all c_j^1 and c_j^2 and α_j^1 and α_j^2.

The fact that $c_j^1 < c_j^2$, as required by the definition of L_j, follows from (6), (7), and the convexity of F_j. Indeed, since F_j is convex, we have

$$c_j^1(x_j - x_j^0) + F_j(x_j^0) > F_j(x_j), \quad \forall x_j \in [\alpha_j^1, x_j^0),$$

and

$$c_j^2(x_j - x_j^0) + F_j(x_j^0) > F_j(x_j), \quad \forall x_j \in (x_j^0, \alpha_j^2].$$

Let $\theta > 0$ be such that

$$x_j^0 - \theta \in [\alpha_j^1, x_j^0), \quad x_j^0 + \theta \in (x_j^0, \alpha_j^2].$$

Then, from the above two inequalities applied to $x_j = x_j^0 - \theta$ and $x_j = x_j^0 + \theta$, respectively, we have

$$-c_j^1\theta + F_j(x_j^0) > F_j(x_j^0 - \theta)$$

and

$$c_j^2\theta + F_j(x_j^0) > F_j(x_j^0 + \theta).$$

Then

$$(c_j^2 - c_j^1)\theta > F_j(x_j^0 + \theta) + F_j(x_j^0 - \theta) - 2F_j(x_j^0) \geq 0.$$

The last inequality follows from the convexity of F_j and implies $c_j^2 > c_j^1$. So we have proved the existence of the functions L_j with the required properties and provided a procedure for constructing L_j.

■

Since $c_j^1 < c_j^2$, it follows that

$$\forall x_j \in (-\infty, x_j^0] : L_j(x_j) = c_j^1(x_j - x_j^0) + F_j(x_j^0) = c_j^1 \min\{x_j - x_j^0, 0\} + F_j(x_j^0),$$

and

$$\forall x_j \in [x_j^0, \infty) : L_j(x_j) = c_j^2(x_j - x_j^0) + F_j(x_j^0) = c_j^2 \max\{x_j - x_j^0, 0\} + F_j(x_j^0).$$

Denote

$$c^1 = (c_1^1, \ldots, c_n^1)^T, \; c^2 = (c_1^2, \ldots, c_n^2)^T.$$

Since at most one of the values $\min\{x_j - x_j^0, 0\}$ and $\max\{x_j - x_j^0, 0\}$ can be nonzero for a given x_j, we can write L as follows:

$$L(x) = (c^1)^T \min(x - x^0, \mathbf{0}) + (c^2)^T \max(x - x^0, \mathbf{0}) + F(x^0), \tag{19}$$

where max and min, applied to pairs of vectors, denote their componentwise maximum and minimum.

Let $I^{\#}(x^0, \delta)$ denote the following instance of the separable convex problem:

$$\begin{aligned} \min\ & L(x) \\ \text{s.t. } & Ax = b, \\ & \mathbf{0} \le x \le u. \end{aligned}$$

It is well known that any convex problem with a piecewise-linear objective function can be formulated as a linear problem where the number of variables is equal to the total number of linear pieces required to describe the convex objective function; e.g., see Dantzig [7]. Therefore, since the functions L_j are piecewise linear and convex, $I^{\#}(x^0, \delta)$ can be formulated as the linear program

$$\begin{aligned} \min\ & (c^1)^T z^1 + (c^2)^T z^2 + F(x^0) - (c^1)^T x^0 \\ \text{s.t. } & A(z^1 + z^2) = b, \\ & \mathbf{0} \le z^1 \le x^0, \\ & \mathbf{0} \le z^2 \le u - x^0. \end{aligned} \tag{20}$$

To prove that the optimal value of this linear program is equal to the optimal value of $I^{\#}(x^0, \delta)$, we use the fact that $c^1_j < c^2_j$, which implies that if $z^2_j > 0$ and $z^1_j < x^0_j$ for a feasible solution $(z^1; z^2)$ then this solution can be improved by adding a positive value to z^1_j and subtracting the same positive value from z^2_j. A sequence of such interchange operations leads from $(z^1; z^2)$ to a new feasible solution $(\min(z^1 + z^2, x^0); \max(z^1 + z^2 - x^0, \mathbf{0}))$, of the linear program (20), with the objective value not greater than the objective value of $(z^1; z^2)$. That is, denoting the objective value of $(z^1; z^2)$ by ξ and taking into account (19), we have

$$\begin{aligned} \xi &= (c^1)^T z^1 + (c^2)^T z^2 + F(x^0) - (c^1)^T x^0 \\ &\ge (c^1)^T \min(z^1 + z^2, x^0) + (c^2)^T \max(z^1 + z^2 - x^0, \mathbf{0}) + F(x^0) - (c^1)^T x^0 \\ &= (c^1)^T \min(z^1 + z^2 - x^0, \mathbf{0}) + (c^2)^T \max(z^1 + z^2 - x^0, \mathbf{0}) + F(x^0) \\ &= L(z^1 + z^2). \end{aligned} \tag{21}$$

The inequalities (21) show that the value of L at $z^1 + z^2$, which is a feasible solution of $I^{\#}(x^0, \delta)$, is not greater than ξ. On the other hand, any feasible solution x of $I^{\#}(x^0, \delta)$ yields a feasible solution $(\min(x, x^0); \max(x - x^0, \mathbf{0}))$ of (20) with the objective value equal to $L(x)$.

So we have proved that the optimal values of $I^{\#}(x^0, \delta)$ and (20) are equal and if $(z^1; z^2)$ is feasible for (20) then $z^1 + z^2$ is a feasible solution of $I^{\#}(x^0, \delta)$ with the objective value not greater than that of $(z^1; z^2)$.

Therefore, x^0 can be improved if and only if the following system is feasible:

$$\begin{aligned} & (c^1)^T z^1 + (c^2)^T z^2 < (c^1)^T x^0 \\ & A(z^1 + z^2) = b, \\ & z^1 \le x^0, \\ & z^2 \ge \mathbf{0}. \end{aligned} \tag{22}$$

Indeed, since $\mathbf{0} < x^0 < u$, any feasible solution $(z^1; z^2)$ of (22) leads to a feasible solution $(x^0 + \lambda(z^1 - x^0); \lambda z^2)$ of (20), for a sufficiently small $\lambda > 0$. The objective value of this solution is equal to

$$\begin{aligned}(c^1)^T(x^0+\lambda(z^1-x^0))+\lambda(c^2)^Tz^2+F(x^0)-(c^1)^Tx^0\\ =\lambda\left((c^1)^T(z^1-x^0)+(c^2)^Tz^2\right)+F(x^0)<F(x^0)=L(x^0).\end{aligned}$$

Then x^0 is not optimal for $I^\#(x^0,\delta)$ because the optimal values of (20) and $I^\#(x^0,\delta)$ are equal. On the other hand, if $I^\#(x^0,\delta)$ has a solution better than x^0, then the linear program (20) has a solution $(z^1;z^2)$ with the objective value less than $L(x^0)$. In this case we have

$$\begin{aligned}(c^1)^Tz^1+(c^2)^Tz^2+F(x^0)-(c^1)^Tx^0<L(x^0)\Rightarrow\\ (c^1)^Tz^1+(c^2)^Tz^2-(c^1)^Tx^0<L(x^0)-F(x^0)=0.\end{aligned}$$

Then $(z^1;z^2)$ is feasible for (22). Thus, finding a feasible descent direction of L at x^0 is equivalent to solving (22).

Lemma 5. *The feasible solution x^0 is not optimal for $I^\#(x^0,\delta)$ if and only if the following homogeneous system is feasible:*

$$\begin{aligned}&-(c^1)^T\bar z^1+(c^2)^T\bar z^2<0,\\ &A(-\bar z^1+\bar z^2)=\mathbf{0},\\ &\bar z^1\geq\mathbf{0},\\ &\bar z^2\geq\mathbf{0}.\end{aligned}\tag{23}$$

If $(\bar z^1;\bar z^2)$ is a feasible solution of (23), then $-\bar z^1+\bar z^2$ is a feasible descent direction of L at x^0.

Proof. The fact that (22) is feasible if and only if (23) is feasible follows from the replacement $z^1=x^0-\bar z^1$ and $z^2=\bar z^2$ (take into account that $Ax^0=b$). Since (22) is feasible if and only if x^0 is not optimal for $I^\#(x^0,\delta)$, it follows that (23) is feasible under exactly the same condition.

Let $(\bar z^1;\bar z^2)$ be a feasible solution of (23). Since $\mathbf{0}<x^0<u$, there is $\lambda>0$ such that $(z^1;z^2)$ with $z^1=x^0-\lambda\bar z^1$ and $z^2=\lambda\bar z^2$ is feasible for (20). That is, we choose a sufficiently small $\lambda>0$ to satisfy $z^1\geq\mathbf{0}$ and $z^2\leq u-x^0$. Then the obtained solution is feasible for (20) because

$$A(z^1+z^2)=Ax^0+\lambda A(-\bar z^1+\bar z^2)=Ax^0=b.$$

Using (21), we can write

$$\begin{aligned}L(x^0+\lambda(-\bar z^1+\bar z^2))&=L(z^1+z^2)\\ &\leq(c^1)^Tz^1+(c^2)^Tz^2+F(x^0)-(c^1)^Tx^0\\ &=-\lambda(c^1)^T\bar z^1+\lambda(c^2)^T\bar z^2+F(x^0)\\ &<F(x^0)=L(x^0).\end{aligned}$$

Since the point $x^0+\lambda(-\bar z^1+\bar z^2)$ is feasible for $I^\#(x^0,\delta)$ and the value of L at this point is strictly less than $L(x^0)$, it follows that $-\bar z^1+\bar z^2$ is a feasible descent direction of L at x^0. ■

Lemma 6. *If x^0 is optimal for $I^\#(x^0,\delta)$, then $F(x^0)\leq OPT(I)+n\delta$.*

Proof. Let x^* be an optimal solution of the original instance I of the separable convex problem. Since the maximum value of the difference $L(x) - F(x)$ does not exceed $n\delta$ for every feasible solution x, which follows from (6) and (7), we have

$$F(x^0) = L(x^0) = OPT(I^{\#}(x^0, \delta)) \leq L(x^*) \leq F(x^*) + n\delta = OPT(I) + n\delta.$$

■

By the definition of α_j^1 and α_j^2, the point x^0 belongs to the interior of the set

(24) $$\mathcal{S} = \{x \in \mathbb{R}^n \mid \alpha_j^1 \leq x_j \leq \alpha_j^2,\ j = 1, \ldots, n\},$$

which is in turn contained in the interior of $\{x \mid \mathbf{0} \leq x \leq u\}$ because both α_j^1 and α_j^2 belong to $(0, u_j)$ for each j. All points x in $\mathcal{S}$ satisfy the condition $F(x) \leq L(x)$. This follows from the convexity of F and the fact that every function L_j is linear in each of the intervals $[0, x_j^0]$ and $[x_j^0, u_j]$ and satisfies (6) and (7), which implies that $F_j(x_j) \leq L_j(x_j)$ for every $x_j \in [\alpha_j^1, \alpha_j^2]$.

The following lemma states that if x is contained in the boundary of $\mathcal{S}$ then the difference between $L(x)$ and $F(x)$ is not less than $\frac{\delta}{2}$.

Lemma 7. *If x belongs to the boundary of $\mathcal{S}$, then $L(x) \geq F(x) + \frac{\delta}{2}$.*

Proof. Since x belongs to the boundary of $\mathcal{S}$, there exists an index j such that x_j is equal to either α_j^1 or α_j^2. It follows that $L_j(x_j) \geq F_j(x_j) + \frac{\delta}{2}$ due to (8) and (9). At the same time, $L_j(x_j) \geq F_j(x_j)$ for the other indices j because x belongs to $\mathcal{S}$. Then $L(x) \geq F(x) + \frac{\delta}{2}$. ■

4. Scaling algorithm.

Assume that we have a polynomial-time algorithm for solving the homogeneous systems of the form (23). We call this algorithm the linear-programming (LP) oracle. This can be any general-purpose polynomial algorithm for linear programming or a suitable specialized algorithm if A has a special structure.

To ensure that the LP oracle runs in polynomial time whenever it is called from our algorithm, we need to guarantee that the sizes of the auxiliary homogeneous systems (23) are polynomially bounded. This is achieved by a suitable transformation of the descent directions found by the oracle in combination with rounding some numbers occurring in the course of the algorithms down to integer multiples of

$$\mu = \frac{\delta}{4nK_{\max}},$$

where $K_{\max}$ is the value defined in (11). From the inequality (12) we conclude that

(25) $$|F(x') - F(x'')| \leq \frac{\delta}{4}$$

for all x' and x'' in $\{x \mid \mathbf{0} \leq x \leq u\}$ with $\|x' - x''\|_\infty \leq \mu$.

The following algorithm finds either an approximate solution to the instance I of the separable convex problem or a solution whose objective value improves the objective value of a given feasible solution x^0, such that $\mathbf{0} < x^0 < u$, by at least $\frac{\delta}{4}$:

Input: $I^{\#}(x^0, \delta)$, where x^0 is a feasible solution with $\mathbf{0} < x^0 < u$;
Output: $x^{\#} = x^0$ if x^0 is optimal for $I^{\#}(x^0, \delta)$. Otherwise, a feasible solution $x^{\#}$ with $\mathbf{0} < x^{\#} < u$ and $F(x^{\#}) \leq F(x^0) - \delta/4$.

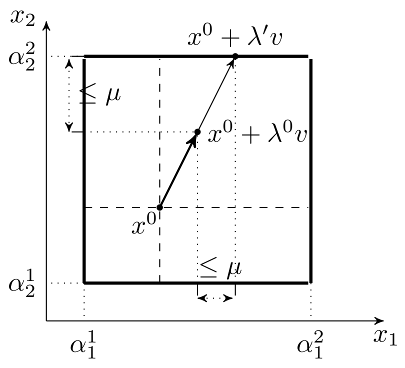


FIGURE 2. *An iteration of Algorithm 4.1 for the case with two variables. The large square depicts the neighborhood $\mathcal{S}$ of x^0. The point $x^0 + \lambda' v$ is the intersection of the ray $\{x^0 + \lambda v | \lambda \geq 0\}$ with $\partial\mathcal{S}$. The gap between L and F in $\partial\mathcal{S}$ is not less than $\delta/2$. The algorithm chooses a suitable approximation λ^0 of λ'. The dashed lines depict the boundaries of the areas where L is linear. In the case with two variables they divide $\mathcal{S}$ into four parts, in each of which L is linear; see the proof of Lemma 9 for details.*

.

```
Call the LP oracle for the respective homogeneous system (23);
**if** the LP oracle returns that (23) is infeasible
**then** decide that $x^0$ is optimal for $I^{\#}(x^0, \delta)$ and set $x^{\#} := x^0$;
**else**
  Using the solution found by the LP oracle, in $O(n^3)$ time find a descent direction $v$
  such that $\|v\|_\infty = 1$ and $size(v)$ is bounded by a polynomial in $size(A)$;
  $\lambda^0 := \max\left\{\mu q |\ x^0 + \mu q \cdot v \in \mathcal{S}, q \in \mathbb{Z}_+\right\}$;
  $x^{\#} := x^0 + \lambda^0 v$;
**end**
Return $x^{\#}$.
```

The reason why we are looking for a descent direction whose size is a bounded by a polynomial in $size(A)$ is that we need a certain control over the sizes of the numbers. A suitable procedure for constructing v by the solution found by the LP oracle is given below. After such a descent direction is found, the above algorithm computes a point in the neighborhood $\mathcal{S}$ of x^0 (defined in (24)) sufficiently close to the intersection of the boundary of $\mathcal{S}$ with the ray $\{x^0 + \lambda v \mid \lambda \geq 0\}$. Again, such an approximate calculation is one of the ingredients that allow us to control the sizes of the numbers. The correctness of the above algorithm is established in Lemma 9. Figure 2 illustrates the algorithm.

REMARK 4.1. *The value λ^0 computed by Algorithm is equal to μq where q is a positive integer with $q \leq \frac{\|u\|_\infty}{\mu}$.*

Proof. The fact that λ^0 is positive follows from Lemma 9, which implies that $F(x^{\#}) < F(x^0)$. The integer q cannot be greater than $\frac{\|u\|_\infty}{\mu}$ because otherwise $x^0 + \mu q$ would have a component violating the constraints $\mathbf{0} \leq x \leq u$. ■

In the above algorithm, to find v, we apply the following procedure. Consider the solution, of the homogeneous system (23), found by the LP oracle. Divide it by the sum of its components. Starting with the obtained solution, in time $O(n^3)$ find a vector being a vertex of the polytope defined by

$$\begin{aligned} &A(-\bar{z}^1+\bar{z}^2)=\mathbf{0},\\ &\mathbf{1}^T(\bar{z}^1+\bar{z}^2)=1,\\ &\bar{z}^1\geq\mathbf{0},\\ &\bar{z}^2\geq\mathbf{0}, \end{aligned} \tag{26}$$

and satisfying the condition

$$-(c^1)^T\bar{z}^1+(c^2)^T\bar{z}^2<0.$$

The vector v is then obtained as

$$v=\frac{1}{\|-\bar{z}^1+\bar{z}^2\|_\infty}(-\bar{z}^1+\bar{z}^2), \tag{27}$$

where $(\bar{z}^1;\bar{z}^2)$ is the respective vertex. The fact that v is a descent direction of L at x^0 follows from Lemma 5.

A suitable vertex $(\bar{z}^1;\bar{z}^2)$ can be constructed by means of the procedure given in the following lemma. Although this procedure is based on a standard argument, we describe it in every detail.

LEMMA 8. *Given a vector c and a solution y of a system $Cx=f, \mathbf{0}\leq x\leq\mathbf{1}$, defining a polytope, where C is a $p\times q$ matrix of rank p, where $p<q$, and f is a vector, we can find a vertex y' of the polytope with $c^Ty'\leq c^Ty$ in time $O(q^3)$.*

Proof. In this proof, by a face we always understand a face of the polytope mentioned in the statement of the lemma.

The procedure that we describe below works as follows. At each iteration, it considers the affine hull of a face containing the current point. In this affine hull, the procedure chooses a direction in which the linear function c^Tx does not increase. Then the procedure moves in this direction as far as the boundary of the polytope (it can happen that the procedure stays in the same point) and finds a constraint $h^Tx\leq\sigma$ which would be violated when moving further in that direction. The respective equation $h^Tx=\sigma$ is then added to the equations defining the affine hull. The dimension of the affine hull of the current face containing the current point is therefore decreased by one. The described procedure is repeated until a vertex is found.

Iteration of the procedure. Let B be a matrix of full rank and d be a vector such that the following system implies $Cx=f$ and defines the affine hull of a face containing y:

$$Bx=d. \tag{28}$$

(At the initial iteration, we simply choose $B=C$ and $d=f$.) The following system defines that face:

$$\begin{aligned} &Bx=d,\\ &\mathbf{0}\leq x\leq\mathbf{1}, \end{aligned} \tag{29}$$

If matrix B is square, then y is a vertex because B has full rank. In this case our procedure returns $y'=y$. Otherwise, compute $(BB^T)^{-1}$ and consider

$$\bar{c}=\begin{cases} c-B^T(BB^T)^{-1}Bc, & \text{if this vector is nonzero,}\\ \text{any nonzero column of } I_q-B^T(BB^T)^{-1}B, & \text{otherwise,}\end{cases}$$

where I_q is the $q \times q$ identity matrix. If the first of the above two cases occurs, then $\bar{c}$ is the orthogonal projection of c onto the linear subspace $\{x \in \mathbb{R}^q \mid Bx = \mathbf{0}\}$. In the second case, when the projection of c is zero, $\bar{c}$ is a nonzero column of the projection matrix $I_q - B^T(BB^T)^{-1}B$. (There must be a nonzero column because I_q and $B^T(BB^T)^{-1}B$ are not equal. This follows from the fact that the rank of B is less than q, which means that the rank of $B^T(BB^T)^{-1}B$ is less than q, while I_q has rank q.) Let $y' = y - \lambda\bar{c}$ where λ is the maximum value such that y' is feasible for (29). We have $c^Ty' \le c^Ty$ because $\lambda \ge 0$. (It may happen that $\lambda = 0$. In this case $y' = y$.) Consider an inequality $h^Tx \le \sigma$ of the system $-x \le \mathbf{0}, x \le \mathbf{1}$, satisfied by y' as an equation and such that $h^T\bar{c} < 0$. For some j, either $h = \mathbf{e}_j$ and $\sigma = 1$ or $h = -\mathbf{e}_j$ and $\sigma = 0$, where $\mathbf{e}_j$ is the vector with the j-th component equal to 1 and the other components equal to 0.

The coefficient matrix of the system

$$\begin{aligned} Bx &= d, \\ h^Tx &= \sigma, \end{aligned} \tag{30}$$

has full rank. To prove this, assume the contrary. Then, since B has full rank and the rank of the coefficient matrix of (30) is equal the rank of the augmented coefficient matrix of (30) due to the feasibility of this system, we conclude that the equation $h^Tx = \sigma$ must be implied by $Bx = d$. Then $h^T\bar{c} = 0$ and we have a contradiction because $h^T\bar{c} < 0$.

The system (30) defines the affine hull of a face containing y'. Consider the equivalent system

$$\begin{aligned} \bar{B}x &= \bar{d}, \\ h^Tx &= \sigma, \end{aligned} \tag{31}$$

where $\bar{B}$ is obtained from B by setting the j-th column to zero and $\bar{d}$ is defined as

$$\bar{d} = d - \sigma B\mathbf{e}_j.$$

The matrix $\bar{B}$ has full rank because the coefficient matrix of the system (31) has full rank. The matrix $\bar{B}\bar{B}^T$ is obtained from BB^T by a rank-1 update; denoting by w the j-th column of B, i.e., $w = B\mathbf{e}_j$, we have

$$\bar{B}\bar{B}^T = BB^T - ww^T.$$

Using the matrix $(BB^T)^{-1}$, we can compute $(\bar{B}\bar{B}^T)^{-1}$ in $O(q^2)$ time by Sherman-Morrison-Woodbury formulas; see Appendix. We have

$$\left(\begin{pmatrix} \bar{B} \\ h^T \end{pmatrix}\begin{pmatrix} \bar{B} \\ h^T \end{pmatrix}^T\right)^{-1} = \begin{pmatrix} (\bar{B}\bar{B}^T) & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}^{-1} = \begin{pmatrix} (\bar{B}\bar{B}^T)^{-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{32}$$

That is, the above matrix (the left-hand side of (32)), which will be used in place of $(BB^T)^{-1}$ at the next iteration, can be computed using the matrix $(BB^T)^{-1}$ in $O(q^2)$ time.

We have constructed a system of inequalities

$$\begin{pmatrix} \bar{B} \\ h^T \end{pmatrix} x = \begin{pmatrix} \bar{d} \\ \sigma \end{pmatrix}, \mathbf{0} \le x \le \mathbf{1}, \tag{33}$$

that defines a face containing y'. The coefficient matrix of the system of equations in (33) has full rank. So we replace (29) by (33) and y by y' and proceed to the next iteration.

Perform the iterations of the described procedure until it returns a vertex y' with $c^T y' \leq c^T y$. The complexity of each iteration except for the initial one is $O(q^2)$. The complexity of the initial iteration is $O(q^3)$ because initially we set $B = C$ and need to compute $(CC^T)^{-1}$. Thus, the required vertex can be found in time $O(q^3)$ because the number of iterations is not greater than q due to the fact that the rank of the coefficient matrix of the equations defining the affine hull of the current face is increased by one with each iteration. ∎

Lemma 9. *Algorithm 4.1 constructs a feasible solution $x^{\#}$ such that*

$$F(x^{\#}) \leq OPT(I) + n\delta$$

or

$$F(x^{\#}) < F(x^0) - \frac{\delta}{4}.$$

Proof. If the LP oracle returns that the homogeneous system (23) is infeasible, then Lemma 5 implies that x^0 is optimal for $I^{\#}(x^0, \delta)$. By Lemma 6 we conclude that

$$F(x^{\#}) \leq OPT(I) + n\delta.$$

Consider the case when x^0 is not optimal for $I^{\#}(x^0, \delta)$. In this case the algorithm finds a descent direction v of L at x^0; see (27). Consider the intersection point

$$x' = x^0 + \lambda' v$$

of the ray $\{x^0 + \lambda v | \lambda \geq 0\}$ with the boundary of $\mathcal{S}$. Function L is linear over this ray because, for every j, the set $\{x_j^0 + \lambda v_j | \lambda \geq 0\}$ is equal to either $(-\infty, x_j^0]$ or $[x_j^0, \infty)$, and, by the definition, each function L_j is linear over both $(-\infty, x_j^0]$ and $[x_j^0, \infty)$. It follows that $L(x') < L(x^0)$ because v is a descent direction of L at x^0. It is clear that x' is a feasible solution of I because $Av = \mathbf{0}$. Since x' lies in the boundary of $\mathcal{S}$, from Lemma 7 it follows that

$$F(x') + \frac{\delta}{2} \leq L(x') < L(x^0) = F(x^0),$$

which implies

$$F(x') < F(x^0) - \frac{\delta}{2}.$$

Note that

$$\|x^{\#} - x'\|_\infty = |\lambda^0 - \lambda'| \cdot \|v\|_\infty = |\lambda^0 - \lambda'| \leq \mu.$$

Then from (25) we have

$$F(x^{\#}) \leq F(x') + \frac{\delta}{4}.$$

It follows that

$$F(x^{\#}) < F(x^0) - \frac{\delta}{4}.$$

∎

The following algorithm calls Algorithm 4.1 until an $n\delta$-approximate solution is found.

```
**Input:** A feasible solution $y$, such that $\mathbf{0} < y < u$, and $\delta > 0$.
**Output:** An $n\delta$-approximate solution $x^{\#}$.
$k := 0$;
$x^{(0)} := y$;
**repeat**
    $k := k + 1$;
    Construct $I^{\#}(x^0, \delta)$ with $x^0 = x^{(k-1)}$;
    Call Algorithm 4.1 for $I^{\#}(x^0, \delta)$;
    Let $x^{\#}$ be the obtained solution;
    $x^{(k)} := x^{\#}$;
**until** $x^0$ is optimal for $I^{\#}(x^0, \delta)$
Return $x^{\#}$.
```

LEMMA 10. *Let $x^{\#}$ be the solution returned by Algorithm 4. Then*

$$F(x^{\#}) \leq OPT(I) + n\delta.$$

The algorithm performs at most $O\left(\frac{F(y)-OPT(I)}{\delta}\right)$ iterations.

Proof. Each point $x^{(k)}$ constructed by the algorithm is feasible and satisfies $\mathbf{0} < x^{(k)} < u$ because $x^{(k)}$ is contained in the respective neighborhood $\mathcal{S}$ of x^0 and both α_j^1 and α_j^2, constructed together with the auxiliary instances $I^{\#}(x^0, \delta)$, are contained in $(0, u_j)$ for all j. If the k-th iteration of the loop is the last one, then the current x^0 is optimal for $I^{\#}(x^0, \delta)$ and $x^{\#} = x^0$. In this case, from Lemma 9 it follows that

$$F(x^{\#}) \leq OPT(I) + n\delta.$$

If the k-th iteration is not the last one, then

$$F(x^{(k)}) < F(x^{(k-1)}) - \frac{\delta}{4},$$

which follows from Lemma 9. Since $F(x^{(k)}) \geq OPT(I)$, we obtain the respective bound on the number of iterations of the loop. ■

The following scaling algorithm chooses δ so that Algorithm 4 is called only polynomially many times until the required accuracy is achieved. Recall that we assume that there are feasible solutions x such that $0 < x < u$. If necessary, we can use linear programming to guarantee this. In polynomial time, we can find a feasible solution z, of polynomial size, with $\mathbf{0} < z < u$.

We assume w.l.o.g. that the accuracy ε lies in $(0, 1]$.

```
SCALING ALGORITHM
**Input:** An accuracy $\varepsilon \in (0, 1]$ and a feasible solution $z$, of polynomial size, with $\mathbf{0} < z < u$.
**Output:** An $\varepsilon$-approximate solution $x^{\#}$.
Let $q$ be the minimum nonnegative integer with the property that
Algorithm 4.1 decides that $z$ is optimal for $I^{\#}(z, 2^{q+1})$;
$t := 0$;
$\delta^{(0)} := 2^q$;
$x^{(0)} := z$;
```

```
repeat
    $t := t + 1$;
    Call Algorithm 4 with $y = x^{(t-1)}$ and $\delta = \delta^{(t-1)}$;
    Let $x^{(t)}$ be the obtained solution;
    $\delta^{(t)} := \frac{\delta^{(t-1)}}{2}$;
until $n\delta^{(t-1)} \leq \varepsilon$
Return $x^{\#} = x^{(t)}$.
```

Denote

$$\Gamma = F(z) - OPT(I).$$

This is the gap between the objective value of the initial solution z and the optimal value. Note that we do not need to know any information about Γ. This value is used only in the complexity estimate of the algorithm.

LEMMA 11. *The scaling algorithm finds an ε-approximate solution $x^{\#}$ in*

$$O\left(\log \frac{n \max\{1, \Gamma\}}{\varepsilon}\right)$$

iterations, each requiring at most $O(n)$ iterations of Algorithm 4.

Proof. The scaling algorithm returns an ε-approximate solution. This follows from the fact that if $n\delta^{(t-1)} \leq \varepsilon$ then

$$F(x^{(t)}) \leq OPT(I) + \varepsilon$$

because, by Lemma 10, $x^{(t)}$ is an $n\delta^{(t-1)}$-approximate solution.

Let us estimate the value $\delta^{(0)}$ calculated at the initial step. (The running time of the initial step is analyzed in the proof of Theorem 12.) Lemma 9 implies that

$$F(z) \leq OPT(I) + 2n\delta^{(0)} \tag{34}$$

because z is optimal for $I^{\#}(z, 2\delta^{(0)}) = I^{\#}(z, 2^{q+1})$. If $q = 0$, then $\delta^{(0)} = 1$. If $q \geq 1$, then, when applied to the instance $I^{\#}(z, \delta^{(0)})$, Algorithm 4.1 finds $x^{\#}$ with

$$OPT(I) \leq F(x^{\#}) < F(z) - \frac{\delta^{(0)}}{4}, \tag{35}$$

which follows from Lemma 9. The inequalities (34) and (35) give us a lower and an upper bound on $\delta^{(0)}$:

$$\frac{\Gamma}{2n} \leq \delta^{(0)} \leq \max\{4\Gamma, 1\}.$$

(The 1 in the maximum corresponds to the case $q = 0$ with $\delta^{(0)} = 1$.) The upper bound on $\delta^{(0)}$ and the stopping condition of the loop imply that the loop performs at most $O\left(\log \frac{n \max\{1,\Gamma\}}{\varepsilon}\right)$ iterations because $\delta^{(t)}$ is reduced by a factor of 2 with each iteration.

Now we analyze how many iterations are required by Algorithm 4 at each iteration of the scaling algorithm.

Consider iteration $t = 1$ of the loop. Lemma 10 implies that Algorithm 4.1 is called by Algorithm 4 at most $O(n)$ times because $\delta^{(0)} \geq \frac{\Gamma}{2n}$ and $F(x^{(0)}) - OPT(I) = F(z) - OPT(I) = \Gamma$.

Consider iteration t with $t \geq 2$. By Lemma 10, the previous iteration ensures

$$F(x^{(t-1)}) \leq OPT(I) + n\delta^{(t-2)} = OPT(I) + 2n\delta^{(t-1)}.$$

Then Lemma 10 implies that Algorithm 4.1 is called by Algorithm 4 at most $O(n)$ times at the current iteration.

Summarizing, we obtain the required estimates on the number of iterations of the scaling algorithm and the number of iterations of Algorithm 4 per iteration of the scaling algorithm. ∎

Let T denote the maximum running time needed by the LP oracle to solve an auxiliary homogeneous system and P denote the maximum running time needed to compute functions F_j in the course of the scaling algorithm. Recall that $K_{\max}$ defined in (11) is an upper bound on the absolute values of the slopes of the supporting lines to the graphs of the functions F_j over the respective intervals $[0, u_j]$.

THEOREM 12. *The scaling algorithm finds an ε-approximate solution in time*

$$O\left(\left(n^3 + T + P \cdot n \cdot \left(\log \frac{nK_{\max}\|u\|_\infty}{\varepsilon}\right)^2\right) \cdot n \cdot \log \frac{n\max\{1,\Gamma\}}{\varepsilon}\right),$$

the values T, P, and the sizes of the numbers in the course of the algorithm being bounded by a polynomial in $size(I)$ and $size(\varepsilon)$.

Proof. In this proof, whenever we say that a value is polynomially bounded, this means that there is a polynomial $g : \mathbb{R}^2 \to \mathbb{R}$ such that this value is bounded by $g(size(I), size(\varepsilon))$, polynomial g being independent of I.

At the initial step of the scaling algorithm, the minimum nonnegative integer q with the required property can be found in $O(\log\Gamma)$ calls to Algorithm 4.1. This can be done as follows. Let q be the current integer. Suppose Algorithm 4.1 does not return that z is optimal for $I^{\#}(z, 2^{q+1})$. In this case $F(z)$ can be improved by at least 2^{q-1} by Lemma 9. It is clear that $2^{q-1} \leq \Gamma$ because otherwise we obtain an objective value less than the optimum. We increase q by one an apply Algorithm 4.1 again. Since $2^{q-1} \leq \Gamma$ if q is not the required integer, we need $O(\log\Gamma)$ iterations of the described procedure. The complexity of constructing $I^{\#}(z, 2^{q+1})$ is at most $O(n(\log K_{\max}\|u\|_\infty)^2)$ by Lemma 4, the sizes of the intermediate numbers in the course of the respective procedure being bounded by a polynomial in $size(K_{\max})$, $size(z)$, and q.

Lemma 11 implies that Algorithm 4.1 is called in total at most $O\left(n \log \frac{n\max\{1,\Gamma\}}{\varepsilon}\right)$ times. Whenever Algorithm 4.1 is called, it runs in time $O\left(n^3 + T\right)$, where $O(n^3)$ stands for the running time of the procedure described in Lemma 8, which we need to guarantee polynomial sizes of the descent directions. The T, defined just before the current lemma, is an upper bound on the running time of the LP oracle. Since $\delta \geq \frac{\varepsilon}{2n}$ whenever Algorithm 4 is called from the scaling algorithm, the construction of the respective instance $I^{\#}(x^0, \delta)$ before each call to Algorithm 4.1 requires $O\left(P \cdot n \cdot \left(\log \frac{nK_{\max}\|u\|_\infty}{\varepsilon}\right)^2\right)$ time; see Lemma 4. Summarizing, we obtain the required running time.

It remains to prove that T, P, and the sizes of all the numbers computed in the course of the algorithm are polynomially bounded.

First, notice that, since the functions F_j are polynomially computable, $size(K_{\max})$ is bounded by a polynomial in $size(I)$. More precisely, using the notation we have introduced in Section 2, we can write that

$$size(K_{\max}) \le 2 \cdot \max_j p(\gamma, size(w(F_j))),$$

where p is a polynomial, $w(F_j)$ denotes the vector encoding the additional information for F_j, and

$$\gamma = \max\{size(0), size(-1), size(u_j + 1), size(u_j)\}.$$

Now we prove that the sizes of the points constructed by the algorithm are polynomially bounded. Let z denote the initial point. At each iteration of Algorithm 4, whenever it is called from the scaling algorithm, we have

$$x^{\#} = z + \sum_{l=1}^{r} \mu_l q_l v^l,$$

where z is the initial point, r is the total number of points constructed, μ_l are the values of μ at the respective calls to Algorithm 4.1 (the values μ_l form a nonincreasing sequence because δ is decreased by the scaling algorithm after each call to Algorithm 4), q_l are positive integers not greater than $\frac{\|u\|_\infty}{\mu_l}$ (see Remark 4.1), and v^l are the respective descent directions found by Algorithm 4.1. Since v^l are obtained from the vertices of the polytope defined by (26) in the way specified by (27), the sizes of v^l are polynomially bounded in the size of A. The sizes of μ_l are polynomially bounded because $size(K_{\max})$ and $size(\delta)$ are polynomially bounded. Then the sizes of q_l are polynomially bounded. It is well known that the size of the sum of a finite number of rational vectors is not greater than two times the sum of the sizes of these vectors. It follows that

$$size(x^{\#}) \le 2\left(size(z) + \sum_{l=1}^{r}\left(size(\mu_l) + size(q_l) + size(v^l)\right)\right), \tag{36}$$

which means that $size(x^{\#})$ is bounded by a polynomial in r, $size(I)$, and $size(\varepsilon)$. Now, to show that $size(x^{\#})$ is polynomially bounded in the course of the algorithm, it remains to prove that the number of iterations is polynomially bounded.

For any optimal solution x^*, we have $\|z - x^*\|_1 \le \|u\|_1$, where z is the initial solution and $\|\cdot\|_1$ denotes the 1-norm. Then the inequality (12) implies that $\Gamma \le K_{\max}\|u\|_1$. It follows that $\log\Gamma$ is bounded by $O(size(K_{\max}) + size(I))$. Therefore, the total number of calls to Algorithm 4.1 is polynomially bounded. Then (36) implies that $size(x^{\#})$ is polynomially bounded in the course of the algorithm. Then it follows from Lemma 4 that the sizes of the auxiliary homogeneous systems (23) solved by the LP oracle and the sizes of α_j^1 and α_j^2 are polynomially bounded. This implies that T is polynomially bounded.

Thus, the sizes of all numbers in the course of the algorithm and the values T and P are polynomially bounded. It follows that the running time of the scaling algorithm is polynomially bounded. ■

So the scaling algorithm is polynomial, provided that we use a polynomial algorithm for linear programming. The corollary below considers a special case in which we can find *exact* optimal solutions in polynomial time.

Corollary 13. *Let the separable convex problem contain a class of instances I such that functions F_j are linear in each segment $[(l-1)s, ls]$, $l \in \mathbb{Z}$, where s is a rational value depending on I such that $size(s)$ is bounded by a polynomial in $size(I)$.*

If s is computable in polynomial time for each instance of the class, then each instance I of the class can be solved in a polynomial number of calls to the scaling algorithm, each call being applied to find an ε-approximate solution to I, for some ε with $size(\varepsilon)$ bounded by a polynomial in $size(I)$.

Proof. Let I be an instance of the mentioned class and s be the respective rational value. Let ε be some given accuracy. Consider the ε-approximate solution $x^{\#}$ returned by the scaling algorithm. Let l_j, $j = 1, \ldots, n$, be such that $x^{\#}_j \in [(l_j - 1)s, l_j s]$ for all j. The sizes of l_j are bounded by a polynomial in $size(s)$ and $size(u)$.

By Lemma 8, in $O(n^3)$ time we can modify $x^{\#}$ into a vertex $x^{\#\#}$ of the polytope

$$\{x \in \mathbb{R}^n | Ax = b, x_j \in [\max\{(l_j - 1)s, 0\}, \min\{l_j s, u_j\}], j = 1, \ldots, n\}$$

such that $F(x^{\#\#}) \leq F(x^{\#})$. (We can apply the lemma because F is linear over this polytope.)

Consider the family $\mathcal{Q}$ of all nonempty polytopes of the form

$$\{x \in \mathbb{R}^n | Ax = b, x_j \in [\max\{(l^*_j - 1)s, 0\}, \min\{l^*_j s, u_j\}], j = 1, \ldots, n\},$$

where $l^*_1, \ldots, l^*_n$, are integers. The sizes of l^*_j are bounded by a polynomial in $size(s)$ and $size(u)$.

If $F(x^{\#\#}) \neq OPT(I)$, then $F(x^{\#\#}) - OPT(I)$ is not less than

$$\min\{F(x) - OPT(I) \mid x \in vert(Q), F(x) > OPT(I), Q \in \mathcal{Q}\}, \tag{37}$$

where $vert(Q)$ denotes the set of vertices of Q.

Let ε^* be (37) divided by two. The size of ε^* is bounded by a polynomial in $size(I)$ because the sizes of the values of F over $vert(Q)$, $Q \in \mathcal{Q}$, are polynomially bounded and the size of $OPT(I)$ is polynomially bounded due to the fact that $F(x^*) = OPT(I)$ for some $x^* \in vert(Q)$ with Q in $\mathcal{Q}$. If $\varepsilon \leq \varepsilon^*$, then the ε-approximate solution $x^{\#\#}$ is an optimal solution of I because otherwise $F(x^{\#\#}) - OPT(I)$ would be not less than the value (37), which is equal to $2\varepsilon^*$.

Consider a convex optimization problem over a convex set S. If a solution is optimal in the intersection of S with a neighborhood of this solution (by a neighborhood of a point we understand a compact set for which this point is interior), then this solution is optimal for the original convex problem. Therefore, to check whether $x^{\#\#}$ is optimal for I, it is sufficient to solve the local problem

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & Ax = b, \\ & \mathbf{0} \leq x \leq u, \\ & x_j \in [x^{\#\#}_j - s, x^{\#\#}_j + s], \forall j. \end{aligned}$$

The last conditions define a neighborhood of $x^{\#\#}$. Since the restriction of F_j to $[x^{\#\#}_j - s, x^{\#\#}_j + s]$ consists of no more than three linear pieces, the above local problem can be formulated as a linear program with at most $3n$ variables. It follows that we can apply any polynomial algorithm for linear programming to verify in polynomial time whether $x^{\#\#}$ is optimal for the above local problem. If $x^{\#\#}$ is not optimal for the local problem, we divide ε by 2 and repeat the described procedure, i.e., we find an ε-approximate solution $x^{\#}$ and perform all the described steps again. If $x^{\#\#}$ is optimal for the local problem, then $x^{\#\#}$ is optimal for I. If the initial ε is equal to 1, then after at most $O(\log \frac{1}{\varepsilon^*})$ iterations of the described procedure we find an optimal solution. (Note that we do not need any information about ε^* in the above procedure. We use ε^* only in the complexity estimate.) ■

Sometimes it is required to find an approximate solution that is close to an exact optimal solution. Following the concept of accuracy introduced by Hochbaum and Shanthikumar [11], we say that a solution x is ρ-accurate if there exists an optimal solution x^* such that

$$\|x - x^*\|_\infty \leq \rho.$$

This means that x is identical to x^* up to $O(\log \frac{1}{\rho})$ significant digits. To find such solutions for separable convex problems, Hochbaum and Shanthikumar [11] propose an algorithm based on linear programming. At each iteration, their algorithm considers an auxiliary linear program with $8n^2\Delta$ variables, where Δ is an upper bound on the the maximum absolute value of the subdeterminants of A. Since Δ is in general exponential in the size of the instance, the algorithm of Hochbaum and Shanthikumar is not polynomial in our case. However, as shown in the proof of the following corollary, we can use one of the main results of [11] and our algorithm to obtain a polynomial running time.

Corollary 14. *The procedure described in Corollary 13 can find a ρ-accurate solution in time which is polynomial in $size(I)$ and $size(\rho)$.*

Proof. Recall that our separable convex problem is encoded by the pair $(\mathcal{F}, w)$ where $\mathcal{F}$ is a class of convex functions of one variable and $w(\varphi)$ denotes the vector containing the additional information needed to compute φ; see the respective definition in Section 2. Consider a rational number s. Let $\tilde{\varphi}$ denote the piecewise linear function coinciding with φ at integer multiples of s. Consider

$$\tilde{\mathcal{F}} = \mathcal{F} \cup \{\tilde{\varphi} \mid \varphi \in \mathcal{F}\}.$$

The only additional information needed to compute $\tilde{\varphi}$ is the number s. Let us define $w(\tilde{\varphi}) = (w(\varphi), s)$ for all φ in $\mathcal{F}$. That is, we have extended w to the whole $\tilde{\mathcal{F}}$. Note that $\tilde{\varphi}$ can be computed in time which is polynomial in $size(w(\tilde{\varphi}))$. We have defined a new separable convex problem $(\tilde{\mathcal{F}}, w)$.

Consider an instance I of $(\mathcal{F}, w)$ and the respective instance $\tilde{I}$ of $(\tilde{\mathcal{F}}, w)$ obtained by replacing each function F_j by the piecewise linear function $\tilde{F}_j$ coinciding with F_j at integer multiples of s. Each system $Ax = b$ with rational coefficients can be represented in polynomial time as a system with integer coefficients. Therefore, without loss of generality we assume that the entries of A are integers. Let us set $\Delta = n!\|A\|_{\max}^n$. This is an upper bound on the absolute values of the subdeterminants of A. Let us choose s as

$$s = \frac{\rho}{2n\Delta}.$$

The size of this value is bounded by a polynomial in $size(A)$ and $size(\rho)$. The proximity theorem presented by Hochbaum and Shanthikumar in [11] (Theorem 3.8 in [11]) implies that every optimal solution x of $\tilde{I}$ satisfies $\|x - x^*\|_\infty \leq \rho$ for some optimal solution x^* of I.

Now find an optimal solution of $\tilde{I}$ using the procedure described in Corollary 13. Since $size(\tilde{I}) = size(I) + n \cdot size(s)$, it follows from the above choice of s that the running time of the procedure in the proof of Corollary 13 is bounded by a polynomial in $size(I)$ and $size(\rho)$. ■

In many cases we are interested in the number of arithmetic operations and evaluations of functions related to the objective function. Let us now relax the condition of polynomial computability. We only assume that there is an oracle being able to compare the values of φ in $\mathcal{F}$ with rational numbers and the values of φ itself.

Corollary 15. *Let K be a known rational upper bound on the absolute values of the slopes of the supporting lines to the graphs of F_j over $[0, u_j]$. Then an ε-approximate solution can be found by the scaling algorithm in*

$$O\left(\left(n^3 + T + n \cdot \left(\log \frac{nK\|u\|_\infty}{\varepsilon}\right)^2\right) \cdot n \cdot \log \frac{n \max\{1, \Gamma\}}{\varepsilon}\right)$$

arithmetic operations and

$$O\left(n^2 \left(\log \frac{nK\|u\|_\infty}{\varepsilon}\right)^2 \log \frac{n \max\{1, \Gamma\}}{\varepsilon}\right)$$

comparisons of the form $\varphi(\lambda) < \varphi(\gamma)$, $\varphi(\lambda) < \gamma$, and $\varphi(\lambda) > \gamma$, where $\varphi \in \mathcal{F}$ and both λ and γ are rationals. (The T and Γ have the same meanings as before.)

Proof. Replace $K_{\max}$ by K. Apply the scaling algorithm. Independently of c^1 and c^2, the descent directions used by the algorithm are rational vectors because A is a rational matrix. Then the feasible solutions constructed by the algorithm are rational vectors. It follows that x^0 is rational in every auxiliary instance $I^{\#}(x^0, \delta)$ constructed in the course of the algorithm. Note that the values of F_j appear only in the comparisons performed by the procedure for maximizing a concave function f with accuracy ρ and in the comparisons performed by the procedures for computing c^1 and c^2. Since x^0 and K are rational, all the numbers in the course of these procedures are rationals. The required complexity follows from Theorem 12. ■

For example, let us consider the separable convex problem

$$\inf \left\{ \sum_{j=1}^{n} c_j x_j + a_j x_j \log_2 x_j \mid Ax = b, \mathbf{0} < x \leq u \right\}, \tag{38}$$

where c_j and a_j are positive rational coefficients and u is a positive integer vector. (The objective functions like that in the above problem arise, for instance, in optimization problems related to the Kullback-Leibler divergence or the relative entropy.) The objective function is convex, but it is defined only for positive vectors. To replace the objective function by a suitable function defined everywhere we use a method similar to that used in Section 2.

Without loss of generality, assume that $\varepsilon < 1$ and

$$\frac{\varepsilon^2}{(4\|a\|_\infty n)^2} = \frac{1}{2^r}$$

for some positive even integer $r \geq 4$. Here, $a = (a_1, \ldots, a_n)^T$. ("W.l.o.g" means that if the above equation was not satisfied, we could find a suitable new value for ε less than the current ε in polynomial time.) Denote

$$\theta = \frac{1}{2^r}.$$

Since $r \geq 4$, we have

$$r \leq 2^{r/2}.$$

Define

$$G_j(x_j) = a_j x_j \log_2 x_j.$$

We can write

$$G_j(\theta) = a_j\theta\log\theta = -\frac{a_j r}{2^r} \geq -\frac{a_j}{2^{r/2}} \geq -\frac{\varepsilon}{4n}. \tag{39}$$

Consider the convex function

$$H_j(x_j) = \begin{cases} G_j(x_j), & x_j \geq \theta, \\ \dfrac{G_j(\theta) - G_j(\theta/2)}{\theta/2}(x_j - \theta) + G_j(\theta), & \text{otherwise.} \end{cases}$$

Using (39) and the fact that $\theta > 0$ and $a_j > 0$, we have

$$-G_j(\theta) + G_j(\theta/2) = -a_j\theta\log_2\theta + a_j\frac{\theta}{2}\log_2\frac{\theta}{2} = -a_j\frac{\theta}{2}\log_2\theta - a_j\frac{\theta}{2} \leq \frac{\varepsilon}{8n}.$$

It follows that, for all $x_j \in [0,\theta]$, since H_j is linear and decreasing in $[0,\theta]$,

$$-\frac{\varepsilon}{4n} \leq G_j(\theta) = H_j(\theta) \leq H_j(x_j) \leq H_j(0) = 2(-G_j(\theta) + G_j(\theta/2)) + G_j(\theta) \leq \frac{\varepsilon}{4n}.$$

(To obtain the last inequality, we also take into account $G_j(\theta) \leq 0$.) Then, taking into account that G_j is decreasing and negative in $(0,\theta]$ and using (39), we conclude that

$$-\frac{\varepsilon}{4n} \leq H_j(x_j) \leq H_j(x_j) - G_j(x_j) \leq H_j(x_j) - G_j(\theta) \leq \frac{\varepsilon}{2n}, \quad \forall x_j \in (0,\theta],$$

which implies

$$|H_j(x_j) - G_j(x_j)| \leq \frac{\varepsilon}{2n}, \quad \forall x_j > 0.$$

Therefore, our choice of θ ensures that the function F defined as

$$F(x) = c^T x + \sum_{j=1}^{n} H_j(x_j)$$

satisfies

$$\left| F(x) - \left(c^T x + \sum_{j=1}^{n} G_j(x_j) \right) \right| \leq \frac{\varepsilon}{2}, \quad \forall x > \mathbf{0}.$$

Then, to solve (38) with the accuracy 2ε, it is sufficient to solve the respective problem of minimizing F with the accuracy ε by means of the scaling algorithm, which delivers an ε-approximate solution $x^{\#}$ with $\mathbf{0} < x^{\#} < u$. The obtained solution is then an 2ε-approximate solution to (38). The comparisons related to $F_j(x_j) = c_j x_j + H_j(x_j)$ reduce to the comparisons of the form $\log_2\lambda < \gamma$ over rational numbers λ and γ. The bound K can be chosen as

$$K = |c_j| + \max\left\{ \frac{G_j(\theta/2) - G_j(\theta)}{\theta/2}, \frac{G_j(2^{\lceil \log_2 \|u\|_\infty \rceil + 1}) - G_j(2^{\lceil \log_2 \|u\|_\infty \rceil})}{2^{\lceil \log_2 \|u\|_\infty \rceil + 1} - 2^{\lceil \log_2 \|u\|_\infty \rceil}} \right\}.$$

The second value in this expression has the form $(G_j(\gamma_2) - G_j(\gamma_1))/(\gamma_2 - \gamma_1)$ where both $G_j(\gamma_1)$ and $G_j(\gamma_2)$ are computable in polynomial time because γ_1 and γ_2 are integer powers of 2 of polynomial size. Moreover, both γ_1 and γ_2 are not less than $\|u\|_\infty$. This value K is computable in time which is polynomial in the size of ε and in the sizes of u and a. So we can apply Corollary 15.

5. A cycle canceling algorithm for network flow problems. Network flow problems with separable convex objective functions can be formulated as separable convex problems over polyhedral sets. Let A be the incidence matrix of a directed graph with m nodes and n arcs. Let b be a vector of supply and demand and u represent capacities of the arcs. Now we can apply the scaling algorithm. Finding a solution of an auxiliary homogeneous system is equivalent to finding a negative-cost cycle in the residual graph whose incidence matrix is $(-A|A)$ and where the costs of the arcs are the respective components of the vector $(-c^1; c^2)$ computed by the procedure that we described in Lemma 4. It is well known (e.g., see Cherkassky and Goldberg [3]) that a negative-cost cycle can be found in time $O(mn)$. Algorithm 4.1 uses a $O(n^3)$-time procedure to ensure that the sizes of the descent directions depend only on the coefficient matrix. Since the sizes of the cycles are polynomially bounded in the dimension, we do not need this procedure anymore. Thus, T in the complexity estimate in Theorem 12 can be replaced by $O(mn)$ and $O(n^3)$ can be omitted. We obtain the running time

$$O\left(\left(nm + P \cdot n \cdot \left(\log \frac{nK_{\max}\|u\|_\infty}{\varepsilon}\right)^2\right) \cdot n \cdot \log \frac{n \max\{1, \Gamma\}}{\varepsilon}\right)$$

for finding an ε-approximate solution.

Since linear functions are separable convex, our algorithm is applicable to the minimum-cost network flow problem with a linear objective function. As an example, let us consider the network depicted in Figure 3. Let all arc capacities u_j be equal to 2 and the cost vector be $c = (4, 1, 1, 4, 1)^T$. Consider $x^0 = \mathbf{1}$. This vector represents a feasible flow. Let $\delta = 1$. Since the objective function is linear, we can construct c^1 and c^2 without use of any special procedure. We simply choose

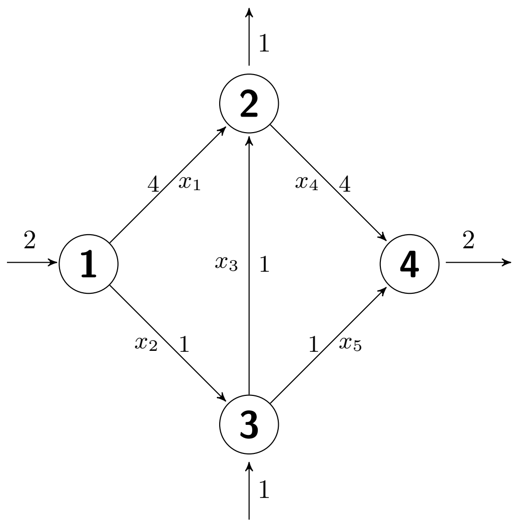
$$c_j^1 = \frac{c_j x_j^0 - \delta}{x_j^0}, c_j^2 = \frac{c_j u_j + \delta - c_j x_j^0}{u_j - x_j^0},$$

and

$$\alpha_j^1 = \frac{x_j^0}{2}, \alpha_j^2 = \frac{x_j^0 + u_j}{2}.$$

Figure 4 explains this choice. All necessary properties can be checked by a direct calculation. Since $x^0 = \mathbf{1}$ and $\delta = 1$ in our example, we have $c^1 = (3, 0, 0, 3, 0)^T$ and $c^2 = (5, 2, 2, 5, 2)^T$. The components of the vectors $-c^1$ and c^2 are depicted as arc costs of the residual graph in Figure 3. The components of $-c^1$ correspond to the reverse arcs. (If we applied the traditional cycle-canceling approach, we had c for the direct arcs and $-c$ for the reverse arcs.) We have several cycles of negative cost. One of them is $((4, 2), (2, 3), (3, 4))$. It corresponds to the descent direction $v = (0, 0, -1, -1, 1)^T$. The parameter μ is equal to $\frac{\delta}{4nK_{\max}} = \frac{1}{80}$ because $n = 5$ and $K_{\max} = 4$. Then, by Algorithm 4.1, $\lambda^0 = 40\mu = \frac{1}{2}$. The vector $x^\#$ is equal to $x^0 + \lambda^0 v = \left(1, 1, \frac{1}{2}, \frac{1}{2}, \frac{3}{2}\right)^T$. We have $c^T x^0 = 11$ and $c^T x^\# = 9$. So, when moving from x^0 to $x^\#$, the objective value is improved by 2 which is greater than $\delta/4 = 1/4$ as proposed by Lemma 9.

Now we give some experimental results for a non-smooth convex network flow



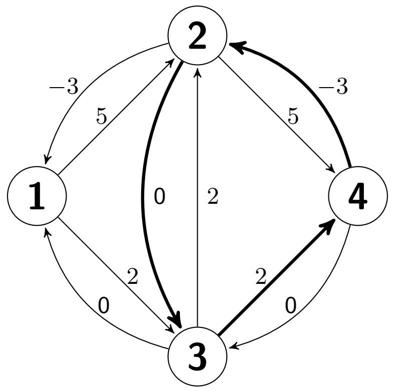


FIGURE 3. *On the left: a network with four nodes, two sources and two sinks. All capacities of the arcs are equal to 2. The numbers at the arcs are costs. On the right: the residual graph with arc costs. The thick arrows form a negative-cost cycle.*

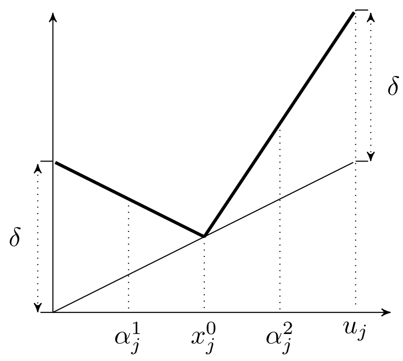


FIGURE 4. *The thick broken line is the graph of L_j. The thin inclined line is the graph of $c_j x_j$. We have chosen $\alpha_j^1 = \frac{x_j^0}{2}$ and $\alpha_j^2 = \frac{x_j^0+u_j}{2}$.*

problem of the form

$$\begin{aligned}\min\ &\sum_{j=1}^{n} a_j\left(\left(\lceil x_j\rceil^2 - \lfloor x_j\rfloor^2\right)(x_j - \lfloor x_j\rfloor) + \lfloor x_j\rfloor^2\right)\\ \text{s.t. } &Ax = b,\\ &\mathbf{0} \le x \le u,\end{aligned}$$

where $a_j > 0$ for all j and an $m \times n$ matrix A is the incidence matrix of a bipartite directed graph where the heads of the arcs belong to the same part of the bipartition and whose underlying undirected graph is $K_{\frac{m}{2},\frac{m}{2}}$. The vector b represents supplies and demands. The u is the vector of capacities. In other words, we consider a convex capacitated transportation problem.

The functions F_j are piecewise-linear convex functions coinciding with $a_j x_j^2$ at

(m,n)	$(10,25)$	$(20,100)$	$(30,225)$	$(40,400)$	$(50,625)$
Avg. num. of linear pieces	12318	50214	115912	197533	310985
Avg. num. of iterations	148	451	939	1673	2599
Avg. time in seconds	14	70	181	371	659
Avg. constraint violation	10^{-12}	10^{-12}	10^{-11}	10^{-11}	10^{-11}

Table 1
Experimental results for random instances.

integer points. Since the rounding can be performed in polynomial time, F_j are polynomially computable.

In place of implementing a combinatorial algorithm for finding negative-cost cycles, we applied the simplex method to the respective linear minimization problems with the constraints (26) and the objective functions of the form $-(c^1)^T\bar{z}^1 + (c^2)^T\bar{z}^2$; the optimal value of a linear problem of this form is negative if and only if the respective homogeneous system (23) is feasible.

The basic feasible solution found by the simplex method with respect to these constraints is the characteristic vector of a negative-cost cycle, in the residual graph, multiplied by a positive scalar, which follows from standard polyhedral properties when we first consider the circulation cone and then observe that the polytope defined by (26) is obtained by intersecting this cone with a hyperplane.

For the implementation we used Matlab running on a standard computer. Note that if $v_j = 0$ in a descent direction v and δ remains the same, then we can use the previous values of c_j^1 and c_j^2. This observation was taken into account in the implementation.

The coefficients a_j were drawn from the uniform distribution over $\{1,\dots,100\}$ and the capacities u_j were drawn from the uniform distribution over $\{1,\dots,1000\}$. The vector b was obtained as $b = \frac{1}{2}Au$ for all the instances.

Table 1 summarizes the experimental results for five sets of instances of the above problem with the numbers of nodes $10, 20, \dots, 50$. Each set contains 10 instances. We choose the accuracy $\varepsilon = 0.001$ for all the instances. The algorithm starts at $z = \frac{1}{2}u$.

The first line of the table indicates the average number of linear pieces obtained as the sum of the values $\mathbf{1}^T u$ over all instances in the respective set, divided by 10, i.e., by the number of instances in the set. This parameter is the average number of variables that would be needed when the instances were formulated as linear programs in a straightforward manner as it was done by Dantzig in [7]. The second line is the average number of calls to the LP oracle. The third line represents the average time needed to solve the instances of the respective set. The fourth line indicates the average maximum constraint violation. (For each instance, the maximum constraint violation is calculated as $\|Ax - b\|_\infty$, where x is the solution obtained by the algorithm.) All the numbers given in the tables are rounded values.

For all the instances, the time spent for constructing auxiliary linear programs was comparable with the time spent to solve these linear programs. Of course, the situation should be different for larger instances. The closeness of the obtained solutions to the boundary, which we calculate as $\min\{\min_j x_j, \min_j u_j - x_j\}$, was within 10^{-9} on average for all the ε-approximate solutions found by the algorithm.

It remains to make a few remarks on separable convex network flow and related problems. Note that replacing each function F_j by a piecewise linear function coinciding with F_j at integer points and using total unimodularity and convexity, and having chosen a suitable value of ε, we can find an exact solution for the case when we additionally require that a feasible flow is integral. In this case, we can perform a polynomial-time procedure, of the type used in Corollary 13, to convert the obtained ε-approximate solution to an integral one, which must be optimal among integral solutions. Of course, specialized algorithms have better performances. For instance, the algorithm by Minoux [17] would need only $O(\log \|u\|_\infty mn^2)$ time to find an integral flow of the minimum cost. The running time of the proximity scaling algorithm of Hochbaum and Shanthikumar [11] is $O(\log \|b\|_\infty m(m + n \log n))$.

Now let us consider a more general problem where A is totally unimodular. For instance, the so-called convex cost integer dual network flow problem can be represented in this form; see Ahuja, Hochbaum, and Orlin [2]. Note that the matrix $(-A|A)$ is totally unimodular as well. Then the problem of finding a feasible descent direction is solvable in $O(n^4)$ time because, in the totally submodular case, an auxiliary homogeneous system is feasible if and only if it has a 0-1 solution, and under this condition the algorithm presented in [4] runs in $O(n^4)$ time. That is, T can be replaced by $O(n^4)$ in that case.

Of course, we can also consider generalized flows, in the same manner, because we do not impose any restrictions on A in the general case.

6. Conclusions. We have developed a polynomial algorithm for the problem of minimizing a polynomially computable separable convex function subject to linear constraints. This result is mostly theoretical because the algorithm uses an LP solver at each iteration, which leads to complexity estimates containing the fourth powers of the number of variables. In its present form, the proposed algorithm can solve instances with about one thousand variables in reasonable time. To increase the performance of the algorithm, one needs to develop a more efficient methodology of using the LP solver. Also, it may be useful to develop faster procedures for constructing the auxiliary LPs in important special cases when we have more information about the objective function.

It would be natural to try to extend our approach to non-separable objective functions. Here, we may follow the general scheme presented at the beginning of Section 3. The question is what conditions must be satisfied by F so that we would be able to construct a suitable function L and formulate the minimization of L as a usual linear program of polynomial size in a higher-dimensional space, analogously to the separable case considered in this article.

Appendix. We begin with a rank-1 update and then apply this special case to a more general situation. Let B be a matrix having full rank and with the number of columns greater than the number of rows. Let $\bar{B}$ be obtained from B by setting the jth column to zero and have full rank as well. We have

$$\bar{B}\bar{B}^T = BB^T - ww^T,$$

where $w = B\mathbf{e}_j$. Here $\mathbf{e}_j$ is the unit vector whose jth component is equal to one. Both matrices BB^T and $\bar{B}\bar{B}^T$ are non-singular. The matrix $\begin{pmatrix} B \\ \mathbf{e}_j^T \end{pmatrix}$ has full rank. Therefore, the matrix

$$\begin{pmatrix} B \\ \mathbf{e}_j^T \end{pmatrix} \begin{pmatrix} B \\ \mathbf{e}_j^T \end{pmatrix}^T = \begin{pmatrix} BB^T & w \\ w^T & 1 \end{pmatrix}$$

is non-singular. Consider the product

$$\begin{pmatrix} BB^T & w \\ w^T & 1 \end{pmatrix} \begin{pmatrix} (BB^T)^{-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} I & w \\ w^T(BB^T)^{-1} & 1 \end{pmatrix}.$$

The matrix at the right-hand side, denote it by M, is non-singular because both matrices at the left-hand side are non-singular. Note that

$$w^T(BB^T)^{-1}w = 1 \Rightarrow w^T(BB^T)^{-1} \begin{pmatrix} I & w \end{pmatrix} = ((w^T(BB^T)^{-1}, 1).$$

Therefore, $1 - w^T(BB^T)^{-1}w \neq 0$ because otherwise there is a sequence of elementary transformations making the last row of M zero. This justifies the application of Sherman-Morrison-Woodbury formulas; see Hager [9]:

$$(\bar{B}\bar{B})^{-1} = (BB^T - ww^T)^{-1} = (BB^T)^{-1} + \frac{(BB^T)^{-1}ww^T(BB^T)^{-1}}{1 - w^T(BB^T)^{-1}w}.$$

Using $(BB^T)^{-1}$, this matrix can be computed in time $O(q^2)$, where q is the number of columns of B.

Acknowledgments. The author is thankful to the associate editor and two anonymous referees for their constructive comments and suggestions.

REFERENCES

[1] Ahuja, R. K. and Orlin, J. B. 2001. Inverse optimization. *Operations Research* **49**, 771-783.

[2] Ahuja R. K., Hochbaum D.S., and Orlin J. B. Solving the convex cost integer dual network flow problem. 2003. *Management Science* **49**, 950-964.

[3] Cherkassky, B.V., and Goldberg, A.V. 1999. Negative-cycle detection algorithms. *Mathematical Programming* **85**, 277-311.

[4] Chubanov, S. 2012. A strongly polynomial algorithm for linear systems having a binary solution. *Mathematical Programming* **134**, 533-570.

[5] Chubanov, S. 2014. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*. DOI 10.1007/s10107-014-0823-8

[6] Chubanov, S. 2014. A strongly polynomial algorithm for linear optimization problems having 0-1 optimal solutions. *Optimization Online*.

[7] Dantzig, G. 1956. Recent Advances in Linear Programming. *Management Science* **2**, 131-144.

[8] Dantzig, G., Johnson, S., and White, W. 1958. A linear programming approach to the chemical equilibrium Problem. *Management Science* **5**, 38-43.

[9] Hager W.W. 1989. Updating the inverse of a matrix. *SIAM Review* **31**, 221-239.

[10] Hochbaum, D. S. 2007. Complexity and algorithms for nonlinear optimization problems. *Annals of Operations Research* **153**, 257-296.

[11] Hochbaum, D. S. and Shanthikumar J. G. 1990. Convex separable optimization is not much harder than linear optimization. *Journal of the Association for Computing Machinery* **37**, 843-862.

[12] Garber, D. and Hazan, E. 2013. A linearly convergent conditional gradient algorithm with appli- cations to online and stochastic optimization. Technical report, arXiv:1301.4666.

[13] Güder, F., and Morris, J.G. 1994. Optimal objective function approximation for separable convex quadratic programming. *Mathematical Programming*, 133-142.

[14] Lan, G. The complexity of large-scale convex programming under a linear optimization oracle. Technical report, arXiv:1309.5550, 2013.

[15] Karzanov, A. and McCormick, Th. 1997. Polynomial methods for separable convex optimization in unimodualr linear spaces with applications. *SIAM Journal on Computing* **26**, 1245-1275.

[16] Meyer, R.R. 1983. Computational aspects of two-segment separable programming. *Mathematical Programming* **26**, 21-39.

[17] Minoux, M. 1986. Solving integer minimum cost flows with separable convex cost objective polynomially. *Mathematical Programming Study* **26**, 237-239.

[18] Monteiro, R. D. C., Adler, I. 1990. An extension of Karmarkar type algorithm to a class of convex separable programming problems with global linear rate of convergence. *Mathematics of Operations Research* **15**, 408 - 422.

[19] Tseng, P., and Bertsekas, D.P. 2000. An ε-relaxation method for separable convex cost generalized network flow problems. *Mathematical Programming* **88**, 85-104.