

# An implementation of steepest-descent augmentation for linear programs

Steffen Borgwardt<sup>1</sup> and Charles Viss<sup>2</sup>

<sup>1</sup> [steffen.borgwardt@ucdenver.edu](mailto:steffen.borgwardt@ucdenver.edu); University of Colorado Denver

<sup>2</sup> [charles.viss@ucdenver.edu](mailto:charles.viss@ucdenver.edu); University of Colorado Denver

**Abstract.** Generalizing the simplex method, circuit augmentation schemes for linear programs follow circuit directions through the interior of the underlying polyhedron. Steepest-descent augmentation is especially promising, but an implementation of the iterative scheme is a significant challenge. We work towards a viable implementation through a model in which a single linear program is updated dynamically to remain in memory throughout. Computational experiments exhibit dramatic improvements over a naïve approach and reveal insight into the next steps required for large-scale computations.

**Keywords:** circuits, linear programming, polyhedra

**MSC:** 52B05, 90C05, 90C08, 90C10

## 1 Introduction

We consider the optimization of a linear objective  $\mathbf{c} \in \mathbb{R}^n$  over a general polyhedron of the form

$$P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, B\mathbf{x} \leq \mathbf{d}\},$$

where  $A \in \mathbb{R}^{m_A \times n}$  and  $B \in \mathbb{R}^{m_B \times n}$ . Recall from [4,5] the definition of the set of *circuits*  $\mathcal{C}(A, B)$  of  $P$ :

**Definition 1 (Circuits).** For a polyhedron  $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, B\mathbf{x} \leq \mathbf{d}\}$ , the set of circuits of  $P$ , denoted  $\mathcal{C}(A, B)$ , consists of those  $\mathbf{g} \in \ker(A) \setminus \{\mathbf{0}\}$  normalized to coprime integer components for which  $B\mathbf{g}$  is support-minimal over the set  $\{B\mathbf{x} : \mathbf{x} \in \ker(A) \setminus \{\mathbf{0}\}\}$ .

Circuits first appeared in the literature as the *elementary vectors* of a subspace [19]. Geometrically, the set of circuits consists of all potential edge directions of  $P$  as the right-hand side vectors  $\mathbf{b}$  and  $\mathbf{d}$  vary [12]. For general purposes, any normalization which results in a unique positive and negative representative for each of these one-dimensional directions can be used when working with circuits. We refer to such a scalar multiple of a circuit  $\mathbf{g} \in \mathcal{C}(A, B)$  as a *circuit direction* of  $P$ . The integer normalization in Definition 1 allows for intuitive interpretations of the circuits of many polyhedra from combinatorial optimization [2,3,6,8,15].

Note that  $\mathcal{C}(A, B)$  is dependent on the representation of a polyhedron. In fact, converting between representations can introduce exponentially many additional directions to the set of circuits [9]. In this paper, we therefore use the above representation for  $P$  which generalizes both standard form and canonical form so that no conversion between representations is required.

As a generalization of the set of edge directions of  $P$ , the set of circuits is a *universal test* set for any linear program over the polyhedron [12]; i.e., given a feasible solution  $\mathbf{x}_0$  to the linear program  $\min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in P\}$ , either  $\mathbf{x}_0$  is an optimal solution or there exists a circuit  $\mathbf{g} \in \mathcal{C}(A, B)$  and a step size  $\alpha > 0$  such that  $\mathbf{x}_0 + \alpha \mathbf{g} \in P$  and  $\mathbf{c}^T \mathbf{g} < 0$ . Circuits are therefore used in the development of *augmentation schemes* for solving linear programs in which

successive, improving, maximal steps are taken along circuit directions until an optimal solution is reached or the problem is determined to be unbounded [10,13,14]. These schemes generalize the simplex method in that they follow circuits, the potential edge directions of the underlying polyhedron; however, their steps are not restricted to only the actual edges of the polyhedron and may traverse its interior. Hence, the *polynomial Hirsch conjecture* – which states that the *combinatorial diameter* of a polyhedron can be bounded polynomially – need not be true in order for there to exist a strongly-polynomial time circuit augmentation scheme for linear programming. For this reason, there has been recent interest in the study of the *circuit diameter* [4,7,15]: the minimum number of circuit steps required to walk from one vertex to another in a polyhedron.

One example of a circuit augmentation algorithm is the *greedy* or *deepest-descent* augmentation scheme from [10,11] which requires at most (weakly) polynomially many steps. The challenge in actually implementing this scheme lies in the computation of the required circuits – a task presumed to be hard. A promising alternative is the *steepest-descent* augmentation scheme of [10] which generalizes the minimum-mean cycle canceling algorithm for solving network flow problems to any bounded linear program [11]. The number of steps needed for this scheme is bounded by the number of circuits [10]. However, a polyhedral model which encodes circuits as vertices can be used to efficiently compute each required circuit [9]. It follows that the algorithm terminates in strongly polynomial time for a polyhedron defined by a totally unimodular matrix [9].

A limitation of this steepest-descent augmentation scheme is that a separate linear program is required at each iteration to provide the circuits. In this paper, we provide an improved implementation in which each iteration requires only a simple change in variable bounds for a single, dynamic linear program. Therefore, the same program remains in memory throughout the algorithm and each of the computed steepest-descent directions serves as a warm-start for the program in the subsequent iteration. Further, we generalize the scheme so that any steepest-descent direction can be used as an augmenting direction – even if it is not necessarily a circuit of the underlying polyhedron. This enables modified implementations in which interior point methods are used to solve the dynamic program.

We begin in Section 2 by formally defining this generalized steepest-descent augmentation scheme and showing that it satisfies the same favorable properties as the original (Theorem 1). Next, in Section 3 we detail our proposed implementation of the scheme. Lastly, computational results are presented in Section 4. We compare various versions of our implementation to each other and to a naïve approach (Section 4.1) and discuss the next steps required for adopting steepest-descent augmentation in a viable algorithm for large-scale computations (Section 4.2).

## 2 Steepest-descent Augmentation

We outline the basic principles behind a steepest-descent augmentation scheme for linear programs. Let  $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, B\mathbf{x} \leq \mathbf{d}\}$  be a general polyhedron and consider the linear program  $LP = \min\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in P\}$ . At iteration  $i$  of an augmentation scheme for solving  $LP$ , assume we have a feasible solution  $\mathbf{x}_i \in P$ . If  $\mathbf{x}_i$  is not optimal, the iteration requires an *augmenting direction*  $\mathbf{y}_i$  such that  $\mathbf{y}_i$  is an *improving direction* with respect to  $\mathbf{c}$  (i.e.,  $\mathbf{c}^T\mathbf{y}_i < 0$ ) and such that  $\mathbf{y}_i$  is *strictly feasible* at  $\mathbf{x}_i$  (i.e., there exists some  $\alpha > 0$  such that  $\mathbf{x}_i + \alpha\mathbf{y}_i \in P$ ). A *maximal step* is then taken in the direction of  $\mathbf{y}_i$  starting at  $\mathbf{x}_i$ : That is,  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{y}_i$  where  $\mathbf{x}_i + \alpha_i\mathbf{y}_i \in P$  but  $\mathbf{x}_i + \alpha\mathbf{y}_i \notin P$  for any  $\alpha > \alpha_i$ .

In the steepest-descent circuit augmentation scheme from [9,10], each of these augmenting directions  $\mathbf{y}_i$  is a so-called *steepest-descent circuit* of  $P$  with respect to  $\mathbf{c}$  at  $\mathbf{x}_i$ :

**Definition 2 (Steepest-descent Circuit).** *Given a polyhedron  $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, B\mathbf{x} \leq \mathbf{d}\}$ , a feasible solution  $\mathbf{x}_0 \in P$ , and an objective  $\mathbf{c} \in \mathbb{R}^n$ , a steepest-descent circuit at  $\mathbf{x}_0$  is a strictly feasible circuit  $\mathbf{g} \in \mathcal{C}(A, B)$  that minimizes  $\mathbf{c}^T\mathbf{g}/\|\mathbf{B}\mathbf{g}\|_1$  over all such circuits.*

It is shown in [9] that such a circuit can be computed by finding a vertex solution to the following LP over a polyhedral model of the set of circuits of  $P$ :

$$\begin{aligned}
& \min \mathbf{c}^T \mathbf{x} \\
& \text{s.t. } A\mathbf{x} = \mathbf{0} \\
& B\mathbf{x} = \mathbf{y}^+ - \mathbf{y}^- \\
& \mathbf{y}_i^+ = 0 \quad \forall i: (B\mathbf{x}_0)_i = \mathbf{d}_i \\
& \sum_{i=1}^{m_B} \mathbf{y}_i^+ + \sum_{i=1}^{m_B} \mathbf{y}_i^- = 1 \\
& \mathbf{y}^+, \mathbf{y}^- \geq \mathbf{0}.
\end{aligned} \tag{model}$$

Using this augmentation scheme, no circuit is ever repeated as an augmenting direction and the algorithm terminates in at most  $|\mathcal{C}(A, B)|$  iterations [9,10]. More specifically, the number of iterations is bounded by the number of different values of  $\mathbf{c}^T \mathbf{g} / \|B\mathbf{g}\|_1$  over all circuits  $\mathbf{g} \in \mathcal{C}(A, B)$  times the dimension of  $P$  – a result known as *Bland's Theorem* [1,10].

When computing an augmenting direction via LP (model), a vertex solution corresponds to a circuit of  $P$  [9]. However, even if the program returns a non-vertex optimal solution (for instance, via an interior point method), we show in Theorem 1 that the corresponding direction can be used in a generalized steepest-descent augmentation scheme for which the above bounds on the number of iterations still hold. Namely, we define a *steepest-descent augmenting direction* at  $\mathbf{x}_0 \in P$  to be any  $\mathbf{y} \in \ker(A)$  which is strictly feasible at  $\mathbf{x}_0$  and minimizes  $\mathbf{c}^T \mathbf{y} / \|B\mathbf{y}\|_1$  over all such directions. A *steepest-descent augmentation* at  $\mathbf{x}_i \in P$  is thus a maximal augmenting step  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{y}_i$  along a steepest-descent augmenting direction  $\mathbf{y}_i$ .

**Theorem 1.** *Consider the linear program  $LP = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in P\}$  where  $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, B\mathbf{x} \leq \mathbf{d}\}$ . Given a feasible solution  $\mathbf{x}_0 \in P$ , the number of steepest-descent augmentations needed to solve  $LP$  is at most*

$$\dim(P) \cdot \left| \left\{ \frac{\mathbf{c}^T \mathbf{g}}{\|B\mathbf{g}\|_1} : \mathbf{g} \in \mathcal{C}(A, B) \right\} \right|.$$

*Proof.* The claim follows from the *conformal sum* property of circuits [12], which states that any  $\mathbf{y} \in \ker(A)$  can be expressed as the sum  $\mathbf{y} = \sum_{j=1}^t \lambda_j \mathbf{g}_j$  of at most  $\dim(P)$  circuits  $\mathbf{g}_j$  of  $P$  (i.e.,  $t \leq n - m_A$ ), where  $\lambda \geq 0$  and each  $B\mathbf{g}_j$  belongs to the same orthant of  $\mathbb{R}^{m_B}$  as  $B\mathbf{y}$ . Therefore, any augmenting direction  $\mathbf{y}$  decomposes into at most  $\dim(P)$  *conformal circuits*  $\mathbf{g}_1, \dots, \mathbf{g}_t$ , where each  $\mathbf{g}_j$  is *sign-compatible* with  $\mathbf{y}$  with respect to the matrix  $B$ .

Let  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_k$  be a sequence of steepest-descent augmentations beginning at  $\mathbf{x}_0$ . We show that for each augmentation  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{y}_i$ , at least one of the conformal circuits of  $\mathbf{y}_i$  must not appear as a conformal circuit in any of the other augmenting directions.

By the definition of steepest-descent augmentations, we retain two useful properties of the (circuit) augmentations from [9,10]: the steepness of consecutive augmenting directions is non-increasing (i.e.,  $-\mathbf{c}^T \mathbf{y}_{i+1} / \|B\mathbf{y}_{i+1}\|_1 \leq -\mathbf{c}^T \mathbf{y}_i / \|B\mathbf{y}_i\|_1$ ) and a change in orthants from  $B\mathbf{y}_i$  to  $B\mathbf{y}_{i+1}$  implies a strict change in the steepness of the steps.

Consider the augmenting direction  $\mathbf{y}_i$ . Assume  $\mathbf{y}_i$  is not a circuit and let  $\mathbf{g}_1, \dots, \mathbf{g}_t$  denote its conformal circuits (i.e.,  $\mathbf{y}_i = \sum_{j=1}^t \lambda_j \mathbf{g}_j$ ). We claim that the steepness  $\mathbf{c}^T \mathbf{y}_i / \|B\mathbf{y}_i\|_1$  of  $\mathbf{y}_i$  is equal to the steepness  $\mathbf{c}^T \mathbf{g}_j / \|B\mathbf{g}_j\|_1$  for each of the  $\mathbf{g}_j$ 's. To see this, note by the definition of conformal circuits that each  $\mathbf{g}_j$  is strictly feasible at  $\mathbf{x}_i$ . Hence, by choice of  $\mathbf{y}_i$ , none of the  $\mathbf{g}_j$ 's is steeper than  $\mathbf{y}_i$ . On the other hand, suppose one of the  $\mathbf{g}_j$ 's is less steep than  $\mathbf{y}_i$ ; i.e.,  $\mathbf{c}^T \mathbf{g}_j / \|B\mathbf{g}_j\|_1 > \mathbf{c}^T \mathbf{y}_i / \|B\mathbf{y}_i\|_1$ . Without loss of generality, assume  $\mathbf{g}_1$  is less

steep than  $\mathbf{y}_1$ . We then have:

$$\begin{aligned}
\mathbf{c}^T(\mathbf{y}_i - \lambda_1 \mathbf{g}_1) &= \mathbf{c}^T \mathbf{y}_i - \lambda_1 \mathbf{c}^T \mathbf{g}_1 \\
&= \|\mathbf{B}\mathbf{y}_i\|_1 \frac{\mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1} - \|\mathbf{B}\mathbf{g}_1\|_1 \frac{\lambda_1 \mathbf{c}^T \mathbf{g}_1}{\|\mathbf{B}\mathbf{g}_1\|_1} \\
&< \|\mathbf{B}\mathbf{y}_i\|_1 \frac{\mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1} - \|\mathbf{B}\mathbf{g}_1\|_1 \frac{\lambda_1 \mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1} \\
&= \frac{(\|\mathbf{B}\mathbf{y}_i\|_1 - \|\mathbf{B}(\lambda_1 \mathbf{g}_1)\|_1) \cdot \mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1} \\
&\leq \frac{(\|\mathbf{B}(\mathbf{y}_i - \lambda_1 \mathbf{g}_1)\|_1) \cdot \mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1},
\end{aligned}$$

where the second inequality follows from the definition of conformal circuits. This implies:

$$\frac{\mathbf{c}^T(\mathbf{y}_i - \lambda_1 \mathbf{g}_1)}{\|\mathbf{B}(\mathbf{y}_i - \lambda_1 \mathbf{g}_1)\|_1} < \frac{\mathbf{c}^T \mathbf{y}_i}{\|\mathbf{B}\mathbf{y}_i\|_1}.$$

However, since  $\mathbf{y}_i - \lambda_1 \mathbf{g}_1$  must itself be strictly feasible at  $\mathbf{x}_i$ , this contradicts the choice of  $\mathbf{y}_i$ . Thus, each  $\mathbf{g}_j$  has the same steepness as  $\mathbf{y}_i$ .

It follows that each of the circuits  $\mathbf{g}_j$  can only appear as a conformal circuit while the steepness of augmenting directions is  $\mathbf{c}^T \mathbf{y}_i / \|\mathbf{B}\mathbf{y}_i\|_1$ . For a fixed steepness value, note that all of the applied augmenting directions must be sign-compatible with each other with respect to  $B$ . Hence, one of the  $\mathbf{g}_j$ 's must not appear as a conformal circuit in any subsequent iteration, else the augmenting step would not have been maximal. Further, at most  $\dim(P)$  augmentations can be applied for a fixed steepness value. To see this, note that due to sign-compatibility, if a maximal step terminates at a facet  $F$  of  $P$ , subsequent augmentations for the current steepness value may not leave  $F$ . Therefore, each step belongs to a face of  $P$  with strictly smaller dimension than that of the previous step.

Since the steepness of augmenting directions throughout the steepest-descent scheme is non-decreasing, it follows that the total number of iterations is bounded by the dimension of  $P$  times the number of different possible steepness values – the bound stated in the theorem.  $\square$

We note the similarity of this steepest-descent augmentation scheme for linear programs to gradient descent for general optimization problems. Gradient descent computes a steepest direction with respect to the Euclidean norm; in the case of linear minimization, this direction is the negative objective projected onto set of strictly feasible directions at the current solution (i.e., the *inner cone* of the current solution). If a solution is on the boundary of the polyhedron, computing this direction is equivalent to the projection of the cost vector onto a polyhedral cone – a quadratic programming problem.

In contrast to the gradient descent direction, computing the steepest-descent direction becomes *easier* as the number of facets containing the current solution increases: Each inequality of  $B\mathbf{x}_0 \leq \mathbf{d}$  which is tight corresponds to a variable which becomes fixed in LP (model). Thus, steepest-descent augmentation can be interpreted as a viable implementation of a *maximal-step* gradient descent scheme for linear programs – instead of using the steepest feasible direction with respect to the Euclidean norm (implicitly assuming that the underlying solution space is a ball in  $\mathbb{R}^n$ ), we compute the steepest feasible direction  $\mathbf{y}$  with respect to the 1-norm of  $B\mathbf{y}$ , which takes into account the combinatorial structure of the underlying polyhedron.

### 3 Implementation

When implementing the steepest-descent augmentation scheme described in Section 2, the constraints of LP (model) change at each iteration according to the active facets at the

current feasible solution. In this section, we modify the program so that only the variable upper bounds change at each iteration. Thus, a dynamic model can remain in memory throughout the algorithm and each of the computed steepest-descent directions can be used to warm-start the model in subsequent iterations.

Note that in LP (model), due to the non-negativity of  $\mathbf{y}^+$ ,  $\mathbf{y}^-$  and the constraint  $\sum_{i=1}^{m_B} \mathbf{y}_i^+ + \sum_{i=1}^{m_B} \mathbf{y}_i^- = 1$ , each of the variables  $\mathbf{y}_i^+$  and  $\mathbf{y}_i^-$  is implicitly bounded above by 1. When the current feasible solution  $\mathbf{x}_0$  satisfies  $(B\mathbf{x}_0)_i = \mathbf{d}_i$ , the upper bound for  $\mathbf{y}_i^+$  is strengthened to 0. Therefore, we can reformulate LP (model) as follows:

$$\begin{aligned}
& \min \mathbf{c}^T \mathbf{x} \\
& \text{s.t. } A\mathbf{x} = \mathbf{0} \\
& B\mathbf{x} - \mathbf{y}^+ + \mathbf{y}^- = \mathbf{0} \\
& \mathbf{y}_i^+ \leq 0 \quad \forall i: (B\mathbf{x}_0)_i = \mathbf{d}_i \\
& \mathbf{y}_i^+ \leq 1 \quad \forall i: (B\mathbf{x}_0)_i < \mathbf{d}_i \quad (\text{steepest}) \\
& \sum_{i=1}^{m_B} \mathbf{y}_i^+ + \sum_{i=1}^{m_B} \mathbf{y}_i^- = 1 \\
& \mathbf{y}^+, \mathbf{y}^- \geq \mathbf{0}.
\end{aligned}$$

Since only the right-hand side of LP (steepest) changes at each iteration, the augmenting direction used in iteration  $i$  of the steepest-descent augmentation scheme corresponds to a dual feasible solution to the program at iteration  $i+1$ . Hence, warm-starting a dual simplex method to solve LP (steepest) at each iteration is a natural approach for computing the required directions. An outline of this proposed implementation of the steepest-descent augmentation scheme is detailed in Algorithm 1.

---

**Algorithm 1** Steepest-descent Augmentation Scheme

---

```

1: procedure STEEPESTDESCENT( $P, \mathbf{c}, \mathbf{x}_0 \in P$ ) ▷ Solves  $LP = \min_{\mathbf{x} \in P} \mathbf{c}^T \mathbf{x}$ 
2:   Initialize LP (steepest) using  $P, \mathbf{c}$ , and  $\mathbf{x}_0$ .
3:    $i \leftarrow 0$ 
4:   Solve LP (steepest) to obtain optimal solution  $(\mathbf{x}^*, \mathbf{y}^+, \mathbf{y}^-)$ 
5:    $\mathbf{y}_i \leftarrow \mathbf{x}^*$ 
6:   while  $\mathbf{c}^T \mathbf{y}_i < 0$  do
7:      $\alpha_i \leftarrow \max\{\alpha \in \mathbb{R}^+ : \mathbf{x}_i + \alpha \mathbf{y}_i \in P\}$ 
8:      $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \alpha_i \mathbf{y}_i$ 
9:     Modify variable upper bounds for LP (steepest) based on  $\mathbf{x}_{i+1}$ 
10:    Using  $\mathbf{y}_i$  as a warm-start, re-solve LP (steepest) to obtain optimal solution  $(\mathbf{x}^*, \mathbf{y}^+, \mathbf{y}^-)$ 
11:     $\mathbf{y}_{i+1} \leftarrow \mathbf{x}^*$ 
12:     $i \leftarrow i + 1$ 
13:   end while
14:   return  $\mathbf{x}_i$ 
15: end procedure

```

---

The success of Algorithm 1 is dependent upon LP (steepest) being easier to solve than the original LP. Note that although LP (steepest) is formulated within a higher-dimensional space than that of the original problem due to the splitting of  $B\mathbf{x}$  into positive and negative parts, the actual increase in dimension of the feasible set is at most  $m_B$  due to the introduced equality constraints. Thus, we can compare this formulation to the introduction of slack variables required when solving the original problem via the simplex method. Additionally, unlike the original problem, LP (steepest) does not depend on the right-hand side vectors  $\mathbf{d}$  or  $\mathbf{b}$ .

Recall also that each active facet at the current solution further reduces the dimension of LP (steepest). Hence, whereas highly degenerate solutions cause inefficiencies in the simplex

method, these points make the computation of a steepest-descent circuit easier. Lastly, note that if  $P$  has a pair of parallel facets, then the corresponding constraints in LP (steepest) are redundant. In particular, if  $B_j = -B_k$ , the program is equivalent to the following:

$$\begin{aligned}
& \min \mathbf{c}^T \mathbf{x} \\
& \text{s.t. } A\mathbf{x} = \mathbf{0} \\
& (B\mathbf{x})_i = \mathbf{y}_i^+ - \mathbf{y}_i^- \quad \forall i \neq k \\
& \mathbf{y}_i^+ = 0 \quad \forall i \neq k: (B\mathbf{x}_0)_i = \mathbf{d}_i \\
& \mathbf{y}_j^- = 0 \quad \text{if } (B\mathbf{x}_0)_k = \mathbf{d}_k \\
& 2\mathbf{y}_j^+ + 2\mathbf{y}_j^- + \sum_{i \neq j, k} (\mathbf{y}_i^+ + \mathbf{y}_i^-) = 1 \\
& \mathbf{y}^+, \mathbf{y}^- \geq \mathbf{0}.
\end{aligned}$$

Taking all of the above into account, an interesting direction of research would be to determine families of LPs for which solving LP (steepest) becomes significantly easier than solving the original problem. However, even without this expert knowledge, we observe from our computational results in Section 4.2 that solving LP (steepest) from scratch is easier than solving the (warm-started) original problem approximately half of the time. When a previous steepest-descent direction is used as a warm-start for LP (steepest), its computation time is faster by an order of magnitude.

Thus, an order-of-magnitude advantage for LP (steepest) is achieved in all but the first iteration of Algorithm 1. There are several ways a potentially effective warm-start could be computed for LP (steepest) in the first iteration as well: A steep, feasible direction at  $\mathbf{x}_0$  could be used to warm-start the primal simplex method (such as the first edge direction chosen by the simplex method when applied to the original problem), or a steep but not necessarily feasible direction could be used to warm-start the dual simplex method (such as the projection of the cost vector  $\mathbf{c}$  onto the affine hull of  $P$ ). A study of these possibilities is another natural direction for future research.

## 4 Computational Results

We implemented the steepest-descent augmentation scheme as outlined in Algorithm 1 using Gurobi [17] to initialize and repeatedly solve LP (steepest) via the dual simplex method. The algorithm was evaluated on a subset of 79 problems from the Netlib LP Test Set [16, 18], a repository which serves as a benchmark for comparing the performance of linear programming algorithms on real-life examples. Code for our experiments is available at <https://github.com/charles-viss/steepest-descent>.

First, in Section 4.1, we evaluate our proposed implementation of the steepest-descent augmentation scheme by comparing it to other alternative implementations; i.e., foregoing the warm-starts at each iteration or using the primal simplex method or an interior point method to repeatedly solve LP (steepest) rather than the dual simplex method. Next, in Section 4.2, we compare the performance of the scheme to that of the simplex method on the original problem and discuss the required next steps towards a viable implementation.

### 4.1 Comparison of Implementations

The average (mean and median) results of the experiments comparing different implementations of the steepest-descent scheme are given in Table 1. We measured the total running time for each variation of the scheme as well as the number of iterations and the time needed to solve LP (steepest) at each iteration. To evaluate the effectiveness of using warm-starts, we record both the average time needed to compute the first steepest-descent direction as well as the average time needed for all steepest-descent computations. All experiments were performed on an Intel i5 8th Gen CPU.

	dual simplex	interior point	primal simplex	dual simplex*
Mean Total Time (s)	<b>3.932</b>	9.243	17.055	18.193
Median Total Time (s)	<b>0.875</b>	3.109	1.814	4.300
Mean Avg. Step Time (ms)	<b>3.78</b>	27.38	25.37	49.30
Median Avg. Step Time (ms)	<b>2.04</b>	18.22	5.89	20.47
Mean First Step Time (ms)	30.20	<b>24.20</b>	33.08	27.50
Median First Step Time (ms)	15.62	15.62	15.62	15.62
Mean SD Iterations	231.6	<b>171.1</b>	234.1	231.6
Median SD Iterations	137.0	<b>119.0</b>	134.5	137.0

Table 1: Comparison of the running time, average step times, and number of iterations for different implementations of the steepest-descent scheme. (\*) indicates the dual simplex method without using warm-starts at each iteration.

We first note the success of the dual simplex implementation of the steepest-descent scheme compared to the primal simplex and interior point implementations. Both total running time and average step times are drastically reduced via dual simplex due to the effectiveness of warm-starts. By comparing the first step times to the average step times, we observe that whereas warm-starts for the interior point method do not appear to have a beneficial impact on average step time, warm-starting the dual or primal simplex method results in significant improvement. For the dual simplex method, warm-starts reduce the average step direction computation time by an order of magnitude. For the primal simplex method, a reduction in computation time is still apparent but not nearly as dramatic – especially when comparing the mean average step times. However, both the interior point and primal simplex implementations (which use warm-starts) significantly outperform the dual simplex implementation when warm-starts are not utilized.

Lastly, we note that the interior point implementation – which is capable of computing augmenting directions that are not necessarily circuits – results in significantly fewer iterations than the other two implementations. This suggests that further improvements to the steepest-descent scheme could be achieved by integrating the capability to compute non-circuit augmenting directions into the proposed dual simplex implementation.

## 4.2 Towards a Viable Implementation

Recall that the steepest-descent augmentation scheme requires an initial feasible solution. To evaluate the viability of the steepest-descent scheme, we therefore compare its performance to that of the primal simplex method warm-started with the same initial solution (i.e., Phase II of the primal simplex method on the original problem). This comparison is detailed in Table 2, which provides the average total running time and number of iterations for both algorithms as well as the first and average step time for the steepest-descent scheme.

We observe that although the *total* running time for the steepest-descent scheme is not competitive with that of the simplex method, computing a first steepest-descent direction via LP ([steepest](#)) is comparable to solving the original problem, while computing subsequent directions is significantly easier. We note also that when computing the first steepest-descent direction, approximately one-third of the time is spent in Phase I of the dual simplex method. On the other hand, the simplex method when applied to the original problem starts immediately at Phase II. We therefore see eliminating Phase I of this initial steepest-descent computation as an important, future angle of attack for improving the steepest-descent scheme.

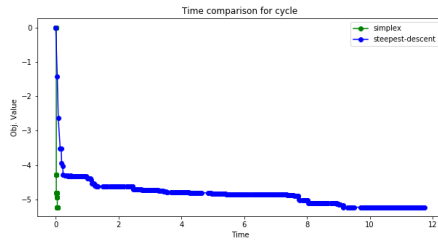
Additionally, we note that although the steepest-descent scheme takes much longer to actually terminate, the decrease in objective value achieved through its first few steps is often similar to that of the simplex method. Consider the results in Figure 1a, which are representative of the behavior of the steepest-descent scheme when applied to many of the



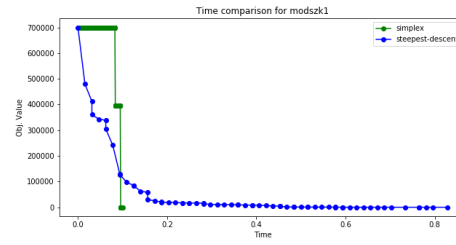
SD Results	Total (s)	N. Iters	First Step (Total, ms)	First Step (Phase I, ms)	First Step (Phase II, ms)	Average Step (ms)
Mean	3.932	231.6	30.20	8.55	21.65	3.78
Median	0.875	137.0	15.62	4.98	10.64	2.04

Simplex Results	Total (ms)	N. Iters
Mean	30.24	452.6
Median	15.61	285.0

Table 2: Breakdown of the total running time, first step time (Phase I and Phase II), and average step time of the steepest-descent (SD) scheme compared to the performance of the simplex method – warm-started with the same starting feasible solution as the SD scheme – on 79 problems from the Netlib LP test set.



(a) Results for `cycle`.



(b) Results for `modszk1`.

Fig. 1: Plots of the objective value over time for the steepest-descent augmentation scheme and the primal simplex method on two problems from Netlib.

Netlib problems. During the first few iterations of the scheme, relatively large steps are taken and the objective value decreases quickly – at a similar rate to that of the simplex method. However, after this fast initial drop, the steepest-descent scheme requires many additional iterations to finally converge to an optimal solution.

Another example which exhibits this behavior is given in Figure 1b. During its first few iterations, the steepest-descent scheme dramatically reduces the objective value while the simplex method stalls at a degenerate vertex. Thus, an optimal strategy for this problem and for similar problems could be to first use the steepest-descent scheme to quickly improve the objective value via large steps along steep, interior directions. Then, once a certain progress threshold is reached, the algorithm could jump to a nearby vertex and terminate quickly via the simplex method. Using such an approach, an optimal solution could potentially be found faster than the time required by either the steepest-descent scheme or the simplex method individually.

Note from Figure 2a that one reason the steepest-descent scheme is initially successful in Figure 1b is the fast computation time for the first steepest-descent direction. In fact, Phase I of this initial computation is virtually nonexistent. Hence, the problem is structured in such a way that LP (steepest) is relatively easy to solve via the dual simplex method – even without warm-starts.

For other problems, this reiterates the need for a warm-start for LP (steepest) at the first iteration. Consider Figures 2b and 2c. The time needed to compute the first steepest-descent direction is almost as much as the time needed for the simplex method to solve the original problem. However, in subsequent iterations when a warm-start is available, the time needed to compute steepest-descent directions is significantly reduced. If the first iteration were to be as fast as subsequent iterations, the steepest-descent scheme may again have an opportunity to outperform the simplex method during its initial iterations.



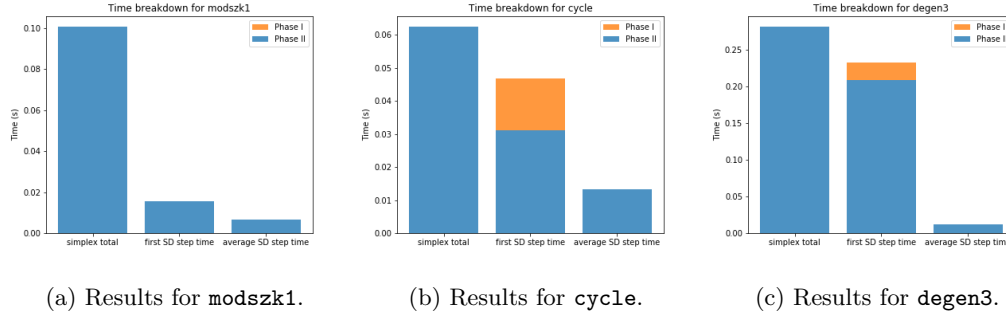


Fig. 2: Solve time for the simplex method compared to the first and average steepest-descent computation times for three Netlib problems.

In summary, further steps are needed for a practically viable implementation of the steepest-descent augmentation scheme. Most importantly, as discussed at the end of Section 3 and as demonstrated by our computational results, a method is needed to quickly compute an effective warm-start for LP (steepest) in the first iteration. Such a direction could be determined based on the underlying LP or computed through tailored algorithms for special families of LPs. There are additional ways in which expert knowledge on the underlying LP could be beneficial: For example, in polyhedra defined by totally unimodular matrices, steps along steepest-descent circuit directions only visit integral solutions [8]. This ensures a lower bound on the decrease in objective value achieved at each iteration. Additionally, it significantly reduces the complexity of computing step sizes and ensures numerical stability – addressing two issues which we observed for several of the Netlib test problems.

## References

1. R. G. Bland and D. L. Jensen. On the computational behavior of a polynomial-time network flow algorithm. *Mathematical Programming, Ser. A*, 54(1):1–39, 1992.
2. S. Borgwardt. On the diameter of partition polytopes and vertex-disjoint cycle cover. *Mathematical Programming, Ser. A*, 141(1):1–20, 2013.
3. S. Borgwardt, J. A. De Loera, E. Finhold, and J. Miller. The hierarchy of circuit diameters and transportation polytopes. *Discrete Applied Mathematics*, 240:8–24, 2018.
4. S. Borgwardt, E. Finhold, and R. Hemmecke. On the circuit diameter of dual transportation polyhedra. *SIAM Journal on Discrete Mathematics*, 29(1):113–121, 2016.
5. S. Borgwardt, E. Finhold, and R. Hemmecke. Quadratic diameter bounds for dual network flow polyhedra. *Mathematical Programming*, 159(1-2, Ser. A):237–251, 2016.
6. S. Borgwardt and F. Happach. Good clusterings have large volume. *Operations Research*, 67(1):215–231, 2019.
7. S. Borgwardt, J. A. De Loera, and E. Finhold. Edges vs circuits: a hierarchy of diameters in polyhedra. *Advances in Geometry*, 16(4):511–530, 2016.
8. S. Borgwardt and C. Viss. Circuit Walks in Integral Polyhedra. *arXiv eprints: 1712.01933v3*, 2017.
9. S. Borgwardt and C. Viss. A polyhedral model for enumeration and optimization over the set of circuits. *Discrete Applied Mathematics*, in print, 2019.
10. J. A. De Loera, R. Hemmecke, and J. Lee. On augmentation algorithms for linear and integer-linear programming: from Edmonds-Karp to Bland and beyond. *SIAM Journal on Optimization*, 25(4):2494–2511, 2015.
11. J. B. Gauthier, J. Desrosiers, and M. Lübbecke. Vector space decomposition for solving large-scale linear programs. *Operations Research*, 66(5):1376–1389, 2018.
12. J. E. Graver. On the foundation of linear and integer programming I. *Mathematical Programming*, 9:207–226, 1975.
13. R. Hemmecke, S. Onn, and L. Romanchuk.  $N$ -fold integer programming in cubic time. *Mathematical Programming, Ser. A*, 137:325–341, 2013.

14. R. Hemmecke, S. Onn, and R. Weismantel. A polynomial oracle-time algorithm for convex integer minimization. *Mathematical Programming, Ser. A*, 126:97–117, 2011.
15. S. Kafer, K. Pashkovich, and L. Sanità. On the circuit diameter of some combinatorial polytopes. *SIAM Journal on Discrete Mathematics*, 33(1):1–25, 2017.
16. T. Koch. The final netlib-lp results. *Operations Research Letters*, 32(2):138–142, 2004.
17. Gurobi Optimization LLC. Gurobi optimizer reference manual, 2019.
18. Netlib. Netlib collection of lp problems in mps format, 2013.
19. R. T. Rockafellar. The elementary vectors of a subspace of  $\mathbb{R}^N$ . In *Combinatorial Mathematics and its Applications*, pages 104–127. University of North Carolina Press, 1969.