On Simplex Pivoting Rules and Complexity Theory

Ilan Adler, Christos Papadimitriou*, and Aviad Rubinstein*

University of California, Berkeley CA 94720, USA

Abstract. We show that there are simplex pivoting rules for which it is PSPACE-complete to tell if a particular basis will appear on the algorithm's path. Such rules cannot be the basis of a strongly polynomial algorithm, unless P = PSPACE. We conjecture that the same can be shown for most known variants of the simplex method. However, we also point out that Dantzig's shadow vertex algorithm has a polynomial path problem. Finally, we discuss in the same context randomized pivoting rules.

Keywords: linear programming, the simplex method, computational complexity.

1 Introduction

Linear programming was famously solved in the late 1940s by Dantzig's simplex method [8]; however, many variants of the simplex method were eventually proved to have exponential worst-case performance [21], while, around the same time, Karp's 1972 paper on NP-completeness [18] mentions linear programming as a rare problem in NP which resists classification as either NP-complete or polynomial-time solvable. Khachiyan's ellipsoid algorithm [20] resolved positively this open question in 1979, but was broadly perceived as a poor competitor to the simplex method. Not long after that, Karmarkar's interior point algorithm [19] provided a practically viable polynomial alternative to the simplex method. However, there was still a sense of dissatisfaction in the community: The number of iterations of both the ellipsoid algorithm and the interior point method depend not just on the dimensions of the problem (the number of variables d and the number of inequalities n) but also on the number of bits needed to represent the numbers in the input; such algorithms are sometimes called "weakly polynomial".

A *strongly polynomial algorithm* for linear programming (or any problem whose input is an array of integers) is one that is a polynomial-time algorithm in the ordinary sense (always stops within a number of steps that is polynomial in the total number of bits in the input), but it also takes a number of elementary arithmetic operations that is polynomial in the dimension of the input array.

* The research of Christos Papadimitriou and Aviad Rubinstein is supported by NSF Grant CCF-0964033.

Strongly polynomial algorithms exist for many network-related special cases of linear programming, as was first shown in [11]. This was extended by Tardos [30] who established the existence of such an algorithm for "combinatorial" linear programs, that is, linear programs whose constraint matrix is 0-1 (or, more generally, contains integers that are at most exponentially large in the dimensions). However, no strongly polynomial algorithm is known for general linear programming.

The following summarizes one of the most important open problems in optimization and the theory of algorithms and complexity:

Conjecture 1. There is a strongly polynomial algorithm for linear programming.

One particularly attractive direction for a positive answer for this conjecture is the search for polynomial variants of the simplex method. It would be wonderful to discover a pivoting rule for the simplex method which (unlike all known such methods) always finds the optimum after a number of iterations that is polynomial in d and n. Hence the following is an interesting speculation:

Conjecture 2. There is a pivoting rule for the simplex method that terminates after a number of iterations that is, in expectation, polynomial in d and n.

In relation to Conjecture 2, clever randomized pivoting rules of a particular recursive sort were discovered rather recently, with worst-case number of iterations that has a subexponential dependence on d [16,23]. Other recent results related to Conjecture 1 can be found in [6,34].

In the next section we formalize the concept of a *pivoting rule:* A method for jumping from one basic solution to an adjacent one that (1) is strongly polynomial per iteration; (2) is guaranteed to increase a potential function at each step; and (3) is guaranteed to always terminate at the optimum (or certify infeasibility or unboundedness). We also give several examples of such rules. It is important to note that in our definition we allow pivoting rules to jump to *infeasible bases* in order to include pivoting rules other than of the primal type. Also, our original definition in Section 2 restricts pivoting rules to be deterministic; we discuss the important subject of randomized rules in Section 5.

Recently there has been a glimmer of hope that some stronger forms of the two conjectures could be disproved, after the disproof of the Hirsch Conjecture [27]. The Hirsch conjecture [9] posited that the diameter of a d-dimensional polytope with n facets is at most $n - d$, the largest known lower bound. The best known upper bound for this diameter is the quasi-polynomial bounds of [17]. But even a super-polynomial lower bound would only falsify the conjectures for *primal* pivoting rules (ones going through only feasible bases, i.e., vertices of the polytope), but *not* for the many other kinds of pivoting rules (see the next section). Furthermore, it is now clear that the techniques involved in the disproof of the Hirsch conjecture are incapable of establishing a nonlinear lower bound on the diameter of polytopes, and it is widely believed that there is a polynomial upper bound on the diameter of polytopes.

In this paper we contemplate whether the concepts and methods of complexity theory can be applied productively to illuminate the problem of strongly polynomial algorithms for linear programming and Conjecture 1. We show a result suggesting that PSPACE-completeness may be relevant.

In particular, we propose to classify deterministic pivoting rules by the complexity of the following problem, which we call THE PATH PROBLEM of a pivoting rule: Given a linear program and a basic solution, will this latter one appear on the pivot rule's path? Recall that PSPACE is the class of problems solvable in polynomial *memory.* This class contains NP, and it is strongly believed to contain it strictly. The PATH PROBLEM of a pivoting rule is clearly in PSPACE, because it can be solved by following the (possibly exponentially long) path of the rule, reusing space; if it is PSPACE-complete, then the pivoting rule cannot be polynomial (unless, of course, P = PSPACE).

But it is not a priori clear that there are pivoting rules for which the path problem is PSPACE-complete. We show (Theorem 1) that they do exist; unfortunately, we prove this not for one of the many classical pivoting rules, but for a new, explicitly constructed — and fairly unnatural — one. We conjecture that the same result holds for essentially all known deterministic pivoting rules; such a proof looks quite challenging; obviously, in such a proof much more will need to be encoded in the linear program (which, in the present proof, is of logarithmic complexity and isomorphic to $\{0,1\}^n$). However, we do exhibit (Theorem 2) a pivoting rule whose path problem is in P: It is Dantzig's well-known *self-dual simplex* [9] (also known as *shadow vertex algorithm*), which is known to be exponential in the worst case [24], but has been used in several sophisticated algorithmic upper bounds for linear programming, such as average-case analysis and smoothness [5,28,2,1,33,29]. We briefly discuss the apparent connection between the average-case performance of a pivoting rule and the complexity of its path problem.

The motivation for our approach came from recent results establishing that it is PSPACE-complete to compute the final result of certain well known algorithms for finding fix points and equilibria [13]. However, the proof techniques used here are completely different from those in [13].

2 Definitions

Consider an algorithm whose input is an array of n integers. The algorithm is called *strongly polynomial* if

- it is polynomial-time as a Turing machine, and
- if one assumes that all elementary arithmetic operations have cost one, the worst-case complexity of the algorithm is bounded by a polynomial in n, and is therefore independent of the size of the input integers.

In linear programming one seeks to maximize $c^T x$ subject to $Ax = b, x \geq 0$, where A is $m \times n$. An $m \times m$ nonsingular submatrix B of A is a *basis.* A *feasible basis* B is one for which the system $Bx_B = b$ (where by x_B we denote vector x

restricted to the coordinates that correspond to B) has a nonnegative solution; in this case, x_B is called a *basic feasible solution.* Basic feasible solutions are important because they render linear programming a combinatorial problem, in that the optimum, if it exists, occurs at one of them. We say that two bases are *adjacent* if they differ in only one column.

There are many versions of linear programming (with inequality constraints, minimization, unrestricted in sign variables, etc.) but they are all known to be easily interreducible. We shall feel free to express linear programs in the most convenient of these.

We shall assume that the linear programs under consideration are non-degenerate (no two bases result in the same basic solution). Detecting this condition is nontrivial (NP-hard, as it turns out). However, there are several reasons why this very convenient assumption is inconsequential. First, a random perturbation of a linear program (obtained, say, by adding a random small vector to b) is non-degenerate with probability one. And second, simplex-like algorithms can typically be modified to essentially perform (deterministic versions of) this perturbation on-line, thus dealing with degeneracy.

We next define a class of algorithms for linear programming that are variants of the simplex method, what we call *pivoting rules.* To start, we recall from linear programming theory three important kinds of bases B, called *terminal bases*:

– optimality: $B^{-1}b \geq 0, c^T - c_B^T B^{-1}A \leq 0$. B is the optimal feasible basis of the linear program.
– unboundedness: $B^{-1}A_j \leq 0, c_j - c_B^T B^{-1}A_j > 0$ for some column A_j of A. This implies that the linear program is unbounded if feasible.
– infeasibility: $(B^{-1})_i A \geq 0, (B^{-1})_i b < 0$ for some row $(B^{-1})_i$ of B^{-1}. This means the linear program is infeasible.

Notice that, given a basis, it can be decided in strongly polynomial time whether it is terminal (and of which kind).

Definition 1. *A* pivoting rule R *is a strongly polynomial algorithm which, given a linear program* (A, b, c)*:*

– *produces an initial basis* B_0*;*
– *given in addition a basis* B *that is not terminal, it produces an adjacent basis* $n_R(B)$ *such that* $\phi_R(n_R(B)) > \phi_R(B)$*, where* ϕ_R *is a potential function.*

The path *of pivoting rule* R *for the linear program* (A, b, c) *is the sequence of bases* $(B_0, n_R(B_0), n_R^2(B_0), \ldots, , n_R^k(B_0))$*, ending at a terminal basis, produced by* R*.*

Obviously, any pivoting rule constitutes a correct algorithm for linear programming, since it will terminate (by monotonicity and finiteness), and can only terminate at a terminal basis. Notice that pivoting rules may pass through infeasible basic solutions (for example, they can start with one). Incidentally, the inclusion of infeasible bases implies that such rules operate not on the linear program's polytope, but on its *linear arrangement.* Since the latter has diameter

$O(mn)$, even the existence of polytopes with super-polynomial diameter will not rule out strongly polynomial pivoting rules.

There are many known deterministic pivoting rules (ties are broken lexicographically, say):

1. **Dantzig's rule (steepest descent).** In this rule (as well as in all other primal rules that follow), given a feasible basis B we first calculate, for each index j not in the basis the objective increase gradient $c_j^B = c_j - c_B^T B^{-1} A_j$. Define $J(B) = \{j : c_j^B > 0\}$. Dantzig's rule selects the $j \in J(B)$ with largest c_j^B and brings it in the basis. By non-degeneracy (if not a terminal basis), this completely determines the next basis. As with all primal pivoting rules, the potential function ϕ_R is the objective.
2. **Steepest edge rule.** Instead of the maximum c_j^B, select the largest $\frac{c_j^B}{||B^{-1}A_j||}$.
3. **Greatest improvement rule.** We bring in the index that results in the largest increment of the objective.
4. **Bland's rule.** Select the smallest $j \in J(B)$.
   For all these rules, however, we have not specified the original basis B_0. This is obviously a problem, since all these rules are primal and need feasible bases, and a feasible basis may not be a priori available. Primal pivoting rules such as these are best applied not on the original $m \times n$ linear program (A, b, c), but to a simple $m \times 2n$ variant called "the big M version," defined as $(A|-A), b, (c|-M, \ldots, -M)$, where M is a large number (M can either be handled symbolically, or be given an appropriate value computed in strongly polynomial time). It is trivial now to find an initial feasible basis. In fact, the pivoting rule running on the new linear program can be thought of as a slightly modified pivoting rule acting on the original linear program (when $j \in J(B)$, A_j is negated, and c_j is replaced by $-M$).
5. **Shadow vertex rule.** Here B_0 is any basis. Given B_0, we construct two vectors c_0 and b_0 such that B_0 is a feasible basis, and also a dual feasible basis, of the relaxed linear program $\max c_0^T x$ subject to $Ax = b_0, x \geq 0$. Now consider the line segment between these two linear programs, with right-hand side and objective $\lambda b + (1-\lambda)b_0$ and $\lambda c + (1-\lambda)c_0$, respectively. Moving on this line segment from $\lambda = 0$, we have both primal-feasible and dual-feasible (and hence optimal) solutions. At some point, one of the two will become infeasible (and only one, by non-degeneracy). We find a new basic solution by exchanging variables as dictated by the violation, and continue. The potential function is the current λ. When $\lambda = 1$ we are at the optimum.
6. **Criss-cross rules.** A class of pivoting rules outside our framework, whose first variant appeared in [35], goes from one (possibly infeasible) basis to the other and convergence to a terminal basis is proved through a combinatorial argument that does not involve an explicit potential function. However, certain such rules (such as the criss-cross pivoting rule suggested in [32]) have been shown ([12]) to possess a monotone potential function, and so they can be expressed within our framework.
7. **Dual pivoting rules.** Naturally, any of the primal pivoting rules can work on the dual.

8. **Primal-dual pivoting rule.** This classical algorithm [10] is an important tool for developing simplex-inspired combinatorial algorithms for a broad set of network problems, acting as a reduction from weighted to unweighted combinatorial problems. It does not conform to our framework, because it involves an inner loop solving a full-fledged linear program.
9. **Pivoting rules with state.** Finally, also outside our framework are pivoting rules relying on data other than A, b, c, and B, for example a pivoting rule relying on statistics of the history of pivoting such as selecting to include the index which has in the past been selected least frequently.
10. **Randomized pivoting rules.** There are several proposed randomized pivoting rules. The ambition here is that the rule's expected path length is polynomial. The simplest one [9] is to pick a random index in $J(B)$. Another important class of randomized rules are the *random facet* rules used in the proofs of subexponential diameter bounds [16,17,23]. We discuss randomized pivoting rules in Section 5.

A pivoting rule is *strongly polynomial* if for any linear program the length of the path is bounded above by a polynomial in m and n. All pivoting rules within our framework mentioned above are known *not* to be strongly polynomial, in that for each one of them there is an explicit family of linear programs with non-polynomial path length, see [3] for a unifying survey.

Explicit constructions are one way of ruling out pivoting rules. *But is there a complexity-theoretic way?* Our interest was sparked by the story of a well-known pivoting rule for a problem other than linear programming: The Lemke-Howson algorithm for two-player Nash equilibrium, discovered in the 1960s [22]. The first explicit construction was obtained decades later [31] and was extremely complicated. More recently, it was established that the problem of finding the Nash equilibrium discovered by the Lemke-Howson algorithm is PSPACE-complete [13] (and thus the algorithm cannot be polynomial, as long as P $\neq$ PSPACE). Remarkably, the PSPACE-completeness proof was much simpler than the explicit construction. We are led to the main definition of this paper:

Definition 2. *The* path problem *associated with a pivoting rule R is the following: Given a linear program and a basis B, does B appear on the path of R for this linear program?*

A pivoting rule is called intractable *if its path problem is PSPACE-complete. A pivoting rule is* tractable *if its path problem can be solved in strongly polynomial time.*

The reason why this concept may be useful in understanding the complexity of linear programming is the following straightforward result:

Proposition 1. *If an intractable pivoting rule is strongly polynomial, then PSPACE = P.*

But are there examples of these two categories? This is the subject of the next two sections.

3 An Intractable Pivoting Rule

This section is devoted to the proof of the following theorem.

Theorem 1. *There is an intractable pivoting rule R.*

The PSPACE-completeness reduction is based on the Klee-Minty construction, the original explicit exponential example for a variant of the simplex method [21], which we recall next.

The *d-dimensional Klee-Minty cube* is the following linear program:

$$\begin{gathered}\max x_1\\ 0 \leq x_d \leq 1\\ \epsilon x_{i+1} \leq x_i \leq 1 - \epsilon x_{i+1}, i = 1, \ldots, d-1\\ x_i \geq 0, i = 1, \ldots, d\end{gathered}$$

The feasible region of this linear program is a distorted d-hypercube (it obviously describes precisely the d-hypercube when $\epsilon = 0$): A polytope whose vertices are within a radius of ϵ from those of a hypercube, and are therefore in one-to-one correspondence with the elements of $\{0,1\}^d$. Thus the feasible bases will also be represented as bit strings in $\{0,1\}^d$. The objective function has a minimum at 0^d (a string of d 0's) and a maximum at 10^{d-1}.

Let us now recall a well-known order on $\{0,1\}^d$ called *Gray code* and denoted G_d. G_1 is simply the order $(0,1)$. Inductively, the Gray code G_{i+1} is $(0G_i, 1G_i^R)$, by which we mean, the sequence G_i with each bit string preceded by a 0, followed by the *reverse* of the order G_i, this time with each bit string preceded by 1. If $0 \leq k < 2^d$, we denote by $G_d[k]$ the k-th bit string in G_d.

G_d is a bijection between $\{0, 1, \ldots, 2^{d-1}\}$ and $\{0,1\}^d$, and therefore we can define the *successor function* $S_d : \{0,1\}^d \mapsto \{0,1\}^d$ as follows: $S_d(x) = G_d[G_d^{-1}(x) + 1]$. The following is straightforward:

Lemma 1. *S_d can be computed in polynomial time.*

Consider a vertex of the Klee-Minty cube of dimension d — equivalently, a bit string $(b_1, \ldots, b_d) \in \{0,1\}^d$. This vertex has d adjacent vertices, each obtained by flipping one of the b_i's. We call the i-th coordinate *increasing* at this vertex if the objective increases by flipping b_i. The following are known important properties of the Klee-Minty cube:

Lemma 2. *(a) The i-th coordinate is increasing if and only if $\sum_{j=1}^{i} b_j$ is even.*
(b) Therefore the sequence of the vertices sorted in increasing objective is precisely G_d.

We next describe the starting PSPACE-complete problem (see e.g., [25] for definitions regarding PSPACE and Boolean circuits).Suppose that we are given a Boolean circuit C with n input bits and n output bits, such that for all inputs x,

x and $C(x)$ always differ in one bit.The *path* of C is the sequence $(x_i, i = 0, \ldots)$, where $x_0 = 0^n$ and $x_{i+1} = C(x_i)$. Consider now this problem: C-PATH: Given C and $x_C \in \{0,1\}^n$, is x_C on the path of C? It is obviously in PSPACE (one need only try the first 2^n bit strings in the path of C, reusing space; if x_C is not reached by that time, we are in a loop and x_C will never be reached). The following is straightforward:

Lemma 3. *There is a family of circuits C of size polynomial in the number of inputs and of polynomial complexity such that C-PATH is PSPACE-complete.*

The reduction proceeds as follows: Given an input $x_C \in \{0,1\}^n$, we shall construct a linear program and a basis $\hat{B}$ such that $\hat{B}$ lies on the path of rule R (yet to be described) if and only if x_C lies on the path of C. The linear program is the Klee-Minty cube of dimension $2n$. The last (least significant) n coordinates of the cube will serve to encode the current bit string on the path of C, while the first n coordinates will maintain a counter in Gray code. We denote the last string of the Gray code, 10^{n-1}, by x_G. The sought basis $\hat{B}$ is taken to be $\hat{B} = x_G x_C$.

Next we describe the pivoting rule R. In fact, it suffices to define R only on Klee-Minty cubes of even dimension — on any other linear program, R can be any pivoting rule, say steepest descent. First, the initial basis of R is $B_0 = 0^{2n}$. Second, here is the description of how R modifies the current basis B (which, since the linear program is the Klee-Minty cube of dimension $2n$ is represented by a bit string of length $2n$):

Pivoting Rule R on basis B:

1. If $B = 10^{2n-1}$, this is a terminal basis and we are done. Otherwise, let $B = (B_1, B_2)$, each a string of length n.
2. If $B_2 = x_C$ then $R(B) = (S_n(B_1), B_2)$.
3. Otherwise, if $B_1 = x_G$ then $R(B) = (B_1, S_n(B_2))$.
4. Otherwise, construct the circuit with n inputs in the family C.
5. Compute $C(B_2)$; suppose that B_2 and $C(B_2)$ differ in the i-th place (by assumption, they only differ in one).
6. If the $n + i$-th coordinate of B is increasing, then $R(B) = (B_1, C(B_2))$.
7. Otherwise, $R(B) = (S_n(B_1), B_2)$.

To explain the workings of R, the first n bits are a counter, and the last n bits encode the current bit string on the path of C from 0^n. If either the first n bits are x_G or the last n bits are x_C, then R just counts up in the other counter (Steps 2 and 3). Otherwise, (Steps 4 and 5), $C(B_2)$ is computed. The intention now is to update the last n bits to be $C(B_2)$. If the flipped coordinate happens to be increasing in B, then this is done immediately (Step 6). But if it is not, then we do the following maneuver: We increment the B_1 counter by flipping the bit in B_1 that leads to the next string in the Gray code (Step 7). This way, in the next invocation of the pivot rule the flipped bit *will* be increasing (by Lemma 2(a)).

To show that R is a pivoting rule, it remains to show that it is strongly polynomial, and that there is a potential function ϕ_R such that the pivot step of R is always monotonically increasing. The former is immediate. For the latter, $\phi_R(B)$ is the value of the objective x_1 in the basic feasible solution represented by B. It is easy to see by inspection of Steps 2, 3, 6, and 7 that in each of these four cases $\phi_R(R(B)) > \phi_R(B)$.

Finally, we must show that $\hat{B}$ is on the path of R if and only if x_C is on the path of C. If x_C is on the path of C then eventually B_2 will be x_C, after at most $2^n - 1$ steps, and from then on Step 2 will be executed to increment the counter B_1. This counter must go through $\hat{B}$ just before arriving at the terminal basis. If x_C is not on the path of C then the path of C will cycle until eventually Step 7 will be executed for a 2^n-th time (it can be easily checked that the cycling of the path of C does not avoid Step 7), at which point $B_1 = x_G$. From then on $\hat{B}$ cannot be reached. This completes the proof of Theorem 1. □

4 A Tractable Pivoting Rule

The pivoting rule we proved intractable is not a natural one. We conjecture that essentially all the pivoting rules described in the last section are intractable (even though proving such a result seems to us challenging). However, here we point out that there *is* a natural, classical pivoting rule that is tractable:

Theorem 2. *The shadow vertex pivoting rule is tractable.*

Proof. Given a linear program (A, b, c), let B_0 be the initial basis, and let b_0 and c_0 be the corresponding initial values of the primal and dual right-hand-side vectors. Given a basis B, we claim that the following is a necessary and sufficient condition that B lies on the path of shadow vertex:

> There is a real number $\lambda \in [0,1]$ such that $(1-\lambda)B^{-1}b_0 + \lambda B^{-1}b \geq 0$ and $(1-\lambda)(c_0^T - (c_0)_B^T B^{-1}A) + \lambda(c^T - c_B^T B^{-1}A) \leq 0$.

In proof, any basis on the path has a non-empty interval of λ's for which these inequalities hold. And if for a given basis B this condition is satisfied, then the inequalities are satisfied for a subinterval of $[0,1]$. If we assume, for contradiction, that B is not on the path of shadow vertex, then we can run the shadow vertex pivoting rule forward and backward from B, and eventually arrive from a different path to the beginning and end, contradicting non-degeneracy. As the condition is a system of $2n$ linear inequalities with one unknown, this completes the proof. □

There is an interesting story here, connecting tractability of pivoting rules and the saga of the average-case analysis of the simplex method. During the early 1980s, and in the wake of the ellipsoid algorithm, average analysis of the simplex method (under some reasonable distribution of linear programs) was an important and timely open question, and indeed there was a flurry of work on that problem [5,28,2,1,33]. It was noticed early by researchers working on this problem that one obstacle in analyzing the average complexity of various versions of the simplex method was a complete inability to predict the path of pivoting rules — that is, the apparent intractability of the path problem we are studying here. And this makes sense: If one cannot characterize well the circumstances under which a vertex will appear on the path, it is difficult to deduce the average performance of the algorithm by adding expectations over all vertices. Once Borgwardt [5] and Smale [28] had the idea of using the shadow vertex pivoting rule in this context, further progress ensued [2,1,33].

5 Randomized Pivoting Rules

Many pivoting rules are explicitly randomized, aiming at good expected performance. Our definition can easily be extended to include randomization: In the definition of a pivoting rule, $R(B)$ is not a single adjacent basis, but a *distribution* on the set of adjacent bases (naturally, this set is polynomially small). Any basis B' in the support of $R(B)$ must satisfy $\phi_R(B') > \phi_R(B)$. Obviously, deterministic pivoting rules are a special case, and therefore Theorems 1 and 2 trivially apply here too.

What is slightly nontrivial is to define what "intractable" means in this case. That is, what is the "path problem" for a randomized pivoting rule R? We believe that the right answer is the following "promise" problem:

Definition 3. *Fix a polynomial p and a function $f : Z^2 \mapsto [0, 1 - \frac{1}{p(m,n)}]$. The* (f,p)-path problem *associated with a randomized pivoting rule R is the following: Given an $m \times n$ linear program and a feasible basis B, distinguish between these two cases: B appears in the execution of R with probability (a) at most $f(m,n)$; and (b) at least $f(m,n) + \frac{1}{p(m,n)}$.*

The analog of Proposition 1 is now:

Proposition 2. *If a randomized pivoting rule R is strongly polynomial in expectation, then the (f,p)-path problem of rule R is in BPP, for all f and p.*

Recall that BPP is the class of all problems that can be solved by randomized algorithms, possibly with a small probability or error, see Chapter 10 in [25].

6 Discussion

Pivoting rules constitute a rich and interesting class of algorithmic objects, and here we focused on one important attribute: whether or not the path problem of a pivoting rule is tractable. We have pointed out that there is an intractable pivoting rule, whereas a well-known classical pivoting rule is tractable. The most important problem we are leaving open is to exhibit a natural intractable pivoting rule. For example, establishing the following would be an important advance:

Conjecture 3. Steepest descent is intractable.

This looks quite challenging. Obviously, in such a proof much more will need to be encoded in the linear program (which, in the present proof, was of logarithmic complexity). The ultimate goal is a generic intractability proof that works for a large class of pivoting rules, thus delimiting the possibilities for a strongly polynomial algorithm. For example: Are all primal pivoting rules (the ones using only feasible bases) intractable?

There are pivoting rules beyond linear programming, usually associated with the linear complementarity problem (LCP, see [7]). They generally do not have a potential function, and termination is proved (when it is proved) by combinatorial arguments. Lemke's algorithm is a well-known general pivoting rule, known to terminate with a solution (or with a certification that no solution exists) in several special cases. It is known to be intractable in general [13], but it can be shown to be tractable when the matrix is positive definite. We conjecture that it is intractable when the matrix is positive semidefinite.

References

1. Adler, I., Karp, R.M., Shamir, R.: A Family of Simplex Variants Solving an m x d Linear Program in Expected Number of Pivot Steps Depending on d Only. Mathematics of Operations Research 11(4), 570–590 (1986)
2. Adler, I., Megiddo, N.: A Simplex Algorithm whose Average Number of Steps is Bounded between Two Quadratic Functions of the Smaller Dimension. Journal of the ACM 32(4), 471–495 (1985)
3. Amenta, N., Ziegler, G.: Deformed products and maximal shadows of polytopes. In: Advances in Discrete and Computational Geometry, pp. 57–90 (1996)
4. Avis, D., Chvatal, C.: Notes on Bland's Pivoting Rule. Math. Programming Study 8, 24–34 (1978)
5. Borgwardt, K.H.: The Average Number of Steps Required by the Simplex Method is Polynomial. Zeitschrift fur Operations Research 26(1), 157–177 (1982)
6. Chubanov, S.: A strongly polynomial algorithm for linear systems having a binary solution. Math. Programming 134(2), 533–570 (2012)
7. Cottle, R., Pang, J.-S., Stone, R.E.: The linear complementarity problem. Academic Press (1992)
8. Dantzig, G.B.: Maximization of a Linear Function of Variables subject to Linear Inequalities (1947); Published in Koopmans, T.C. (ed.): Activity Analysis of Production and Allocation, pp. 339–347. Wiley & Chapman-Hall (1951)
9. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press and the RAND Corporation (1963)
10. Danzig, G.B., Ford, L.R., Fulkerson, D.R.: A Primal-Dual Algorithm for Linear Programming. In: Kuhn, H.W., Tucker, A.W. (eds.) Linear Inequalities and Related Systems, pp. 171–181. Princeton University Press (1954)
11. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19(2), 248–264 (1972)
12. Fukuda, K., Matsui, T.: On the Finiteness of the Criss-cross Method. European Journal of Operations Research 52, 119–124 (1991)

13. Goldberg, P.W., Papadimitriou, C.H., Savani, R.: The Complexity of the Homotopy Method, Equilibrium Selection, and Lemke-Howson Solutions. ACM Transactions on Economics and Computation 1(2), Article 9 (2013)
14. Goldfarb, D., Reid, J.K.: A Practical Steepest-Edge Simplex Algorithm. Mathematical Programming 12, 361–371 (1977)
15. Jeroslow, R.G.: The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement. Discrete Mathematics 4, 367–377 (1973)
16. Kalai, G.: A Subexponential Randomized Simplex Algorithm. In: ACM 24th Symposium on Theory of Computing, pp. 475–482 (1992)
17. Kalai, G., Kleitman, D.: Quasi-polynomial Bounds for the Diameter of Graphs and Polyhedra. Bull. Amer. Math. Soc. 26, 315–316 (1992)
18. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
19. Karmarkar, N.K.: A New Polynomial-time Algorithm for Linear Programming. Combinatorica 4, 373–395 (1984)
20. Khachian, R.M.: A Polynomial Algorithm in Linear Programming. Doklady Akad. Nauk SSSR 244(5), 1093–1096; Translated in Soviet Math. Doklady, 20, 191–194 (1979)
21. Klee, V., Minty, G.J.: How Good is the Simplex Algorithm? In: Shisha, O. (ed.) Inequalities III, pp. 159–175. Academic Press, New York (1972)
22. Lemke, C.E., Howson, J.T.: Equilibrium points of bimatrix games. SIAM Journal on Applied Mathematics 12(2), 413–423 (1996)
23. Matousek, J., Sharir, M., Welzl, E.: A Subexponential Bound for Linear programming. Algorithmica 16(4-5), 498–516 (1996)
24. Murty, K.G.: Computational Complexity of Parametric Linear Programming. Mathematical Programming 19, 213–219 (1980)
25. Papadimitriou, C.: Computational Complexity. Addison Wesley (1994)
26. Ross, C.: An Exponential Example for Terlaky's Pivoting Rule for the Criss-cross Simplex Method. Mathematical Programming 46, 78–94 (1990)
27. Santos, F.: A Counterexample to the Hirsch Conjecture. Annals of Mathematics 176(1), 383–412 (2013)
28. Smale, S.: On the Average Number of Steps of the Simplex Method of Linear Programming. Mathematical Programming 27, 241–267 (1983)
29. Spielman, D.A., Teng, S.-H.: Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. Journal of the ACM 51(3), 385–463 (2004)
30. Tardos, E.: A Strongly Polynomial Algorithm to Solve Combinatorial Linear Programs. Operations Research 34, 250–256 (1986)
31. Savani, R., von Stengel, B.: Hard to Solve Bimatrix Games. Econometrica 74(2), 397–429 (2006)
32. Terlaky, T.: A Convergent Criss-cross Method. Optimization 16, 683–690 (1990)
33. Todd, M.J.: Polynomial Expected Behvior of Pivoting Algorithm for Linear Complementarity and Linear Programming Problems. Mathematical Programming 35, 173–192 (1986)
34. Ye, Y.: The Simplex and Policy-Iteration Methods Are Strongly Polynomial for the Markov Decision Problem with a Fixed Discount Rate. Mathematics of Operations Research 36(4), 593–603 (2011)
35. Zionts, S.: The Criss-cross Method for Solving Linear Programming Problems. Management Science 15, 426–445 (1979)