Fast Algorithms for Monotonic Discounted Linear Programs with Two Variables per Inequality*

Daniel Andersson† and Sergei Vorobyov‡§
Uppsala University, Sweden

May 19, 2006

Abstract

We suggest new strongly polynomial algorithms for solving linear programs $\min(\sum x_i|S)$ with constraints S of the *monotonic discounted* form $x_i \geq \lambda x_j + \beta$ with $0 < \lambda < 1$. The algorithm for the case when the discounting factor λ is equal for all constraints is $O(mn^2)$, whereas the algorithm for the case when λ may vary between the constraints is $O(mn^2 \log m)$, where n is the number of variables and m is the number of constraints.

As applications, we obtain the best currently available algorithm for two-player discounted payoff games and a new faster strongly subexponential algorithm for the ergodic partition problem for mean payoff games.

1 Introduction

One of the most important open problems in mathematical optimization is the existence of a strongly polynomial algorithm for linear programming. In the (weakly) polynomial ellipsoid algorithm due to Khachiyan and the interior-point algorithms of Karmarkar, the number of operations depends not only on the number of variables n and constraints m, but also on the magnitude of the coefficients. The quest for a strongly polynomial algorithm, with a running time polynomial in n, m, while independent of the coefficients, has continued for the last quarter of a century.

Particular cases of this fundamental problem appear in solving infinite games. For instance, the only currently known way to efficiently solve the one-player version of Shapley's stochastic games [10] (or Markov decision processes) is by solving a linear program (with no known strongly polynomial algorithms). However, this program $\min(\sum x_i|S)$ has many special properties. The constraints of S are *monotonic discounted*, i.e., have form $x_i \geq \sum_{i \neq j} a_j x_j + \beta$, with $0 \leq a_j$ and $\sum a_j < 1$. The feasible region contains a unique minimal element, which is also the solution to the program and the game.

*Preprint NI06019-LAA, Isaac Newton Institute for Mathematical Sciences, Cambridge, United Kingdom (`http://www.newton.cam.ac.uk/`).

†Supported by the Emma and Anders Zorn Foundation.

‡Support by the EPSRC Grant 531174 and by the VR Grant 50861101 is gratefully acknowledged.

§Participant of the Logic and Algorithms Programme organized at the Isaac Newton Institute.

A few particular classes of linear programs allowing for strongly polynomial algorithms are known. For example, [4, 7] (see also references therein) give strongly polynomial algorithms for finding *feasible solutions* of arbitrary, not necessarily monotonic, linear programs with two variables per inequality.

In this paper we suggest new strongly polynomial algorithms for solving *to optimality*[1] a particular case of the above monotonic linear programs, consisting of *MD2-constraints* (monotonic discounted 2-variable constraints). An MD2-constraint has the form $x_i \geq \lambda x_j + \beta$ for some β, $\lambda \in \mathbb{R}$ with $0 < \lambda < 1$ (a single variable constraint $x_i \geq \beta$ can be expressed as $x_i \geq \lambda x_i + \beta(1-\lambda)$). An MD2-linear program consists in minimizing the sum of all variables $\sum x_i$ subject to a system of MD2-constraints in which every variable appears in the left-hand side of at least one constraint.[2]

We describe two different algorithms for two kinds of MD2-linear programs. First, in Section 3 we address the case when the discounting factors λ are *equal* for all constraints. The underlying idea of the algorithm seems to be new, although very natural and admitting an elegant runtime analysis. Starting with a feasible solution, which is easy to find for MD2-constraints (linear time in the number of constraints), we consider trees defined by constraints *tightly* satisfied by the current feasible solution, i.e., $x_i = \lambda x_j + \beta$ (taking at most one per left-hand side variable). A variable not occurring on the left in any tight constraints is called a root. Pulling down the root of such a tree, by decreasing the values of all nodes while trying to preserve all tree equalities leads to either the disappearance of the root, or a defection of a subtree to another tree, or an internal switch, when a subtree reconnects in an alternative way within the same tree. Using a greedy strategy results in an $O(mn^2)$ algorithm, and the equality of discounting factors is essential for the analysis, allowing to (tightly) bound the number of internal switches to the square of the initial number of tree nodes. This case covers the standard one-player discounted payoff games. When combined with the randomized techniques for two player games [2] (as a subroutine), it results in the best currently known subexponential algorithm for discounted payoff games. It also yields a new, more efficient, strongly subexponential algorithm for the ergodic partition problem for mean payoff games; see Section 5.

Second, in Section 4 we consider the case of different discounting factors. In this case it is possible to modify the algorithm of [7] for feasibility of two-variable systems of linear constraint for solving MD2-linear programs to optimality. The resulting bound for finding the optimum is $O(mn^2 \log m)$, the same as for the feasibility algorithm [7].

Relation to the Linear Complementarity Problem (LCP). An instance (A, b) of the LCP is given by a square matrix $A \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$, and consists in finding a vector $x \geq 0$ such that $Ax + b \geq 0$ and $x^T(Ax + b) = 0$ [8, 5]. Recall that a square real matrix is called: 1) a Z-matrix if its all off-diagonal elements are nonpositive; 2) a P-matrix if its all principal minors are positive (in particular, all diagonal elements are positive); 3) a K-matrix if it is simultaneously a Z- and a P-matrix. For a P-matrix A the LCP instance (A, b) has a unique solution for every b. Chandrasekaran's algorithm solves instances of the Z-matrix (hence, K-matrix) LCP problem in strongly polynomial time [5]. When A is a K-matrix, the unique solution found by the algorithm coincides with the least element of the feasible set $S = \{x \geq 0, Ax + b \geq 0\}$, or, equivalently, with

[1] Note that feasibility for our systems of constraints can easily be found in $O(m)$ time; see below.

[2] This is needed to guarantee finiteness of the optimum.

the unique optimal solution of the linear program $\min(p^T x|S)$ for any positive vector p. Thus Chandrasekaran's algorithm solves the above linear programs with square K-matrices in strongly polynomial time. Such linear programs are necessarily *monotonic*, i.e., each inequality has the form $x_i \geq p^T x$, with $p \geq 0$, and have *at most two* constraints with each variable on the left (with $x_i \geq 0$ being the second). A more general problem of strongly polynomial solvability of such systems when *more than two* constraints per variable on the left-hand side is open. This problem is equivalent to the so-called K-matrix Generalized LCP. Our paper solves a particular case of this more general problem, when constraints are monotonic, discounted, and contain at most one variable on the right of each constraint (or two variables per constraint).

2 Preliminaries

Throughout the paper we use the standard linear algebraic notation and conventions, e.g., assume that vectors are column vectors denoted by letters x, y, z, a, b, etc. Corresponding indexed letters, like x_i, denote vector coordinates, juxtaposition means vector scalar product, and x^T denotes the transposition of vector x. We always let n denote the dimension of the underlying real vector space $\mathbb{R}^n$ and m the number of linear constraints in the system under consideration. We tacitly assume that $m \geq n$. Depending on the context, 1 can denote the all-ones vector $\langle 1, \ldots, 1\rangle \in \mathbb{R}^n$.

Definition 2.1 (Monotonic Discounted Constraints) *A vector $a \in \mathbb{R}^n$ is called* monotonic discounted *(MD-vector) if it has a unique component equal to 1, all its other components are nonpositive, and sum up to a number strictly greater than -1. A* monotonic discounted constraint *(MD-constraint) has the form $a^T x \geq \beta$ for an MD-vector $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$.*

An MD2-vector is an MD-vector with at most one negative component, and an MD2-constraint has the form $a^T x \geq \beta$ for an MD2-vector $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$.

The i-th group S_i of a system S of MD- or MD2-constraints consists of all constraints $a^T x \geq \beta$ of S in which the vector a has the i-th component equal to 1. A system S is full *if $S_i \neq \emptyset$ for each $i \in \{1, \ldots, n\}$.* □

The feasible region for an MD-system is always non-empty and we can easily find a feasible solution.

Proposition 2.2 *A feasible solution for a given MD-system can be computed in $O(mn)$ time. For an MD2-system, $O(m)$ time suffices.*

Proof. The n-dimensional vector $1\xi = \langle \xi, \ldots, \xi\rangle$ is feasible for an MD-system iff for each constraint $a^T x \geq \beta$ we have $a^T(1\xi) \geq \beta$ or, equivalently, $\xi \geq \beta/(1^T a)$. Thus, taking ξ to be the maximum of $\beta/(1^T a)$ over all constraints, makes 1ξ a feasible solution satisfying at least one constraint as equality. □

For two vectors $x \leq y$ means componentwise $x_i \leq y_i$ for each $i \in \{1, \ldots, n\}$, and $\geq$ is used correspondingly. A vector x is a *least element* of the set X if $x \leq y$ for every $y \in X$. The main problem we concentrate upon in this paper is the following.

MD2-Least Element Problem (MD2-LEP).

Given: a full system S of MD2-constraints.

Find: the least element of the convex polyhedron defined by S. □

This least element always *exists* and is *uniquely defined*, which follows from Proposition 2.3, summarizing well-known elementary properties of MD-systems.

For a full system of MD-constraints S call S' a *representative subsystem* of S if S' contains exactly one constraint of S per group. A *representative equality subsystem* is a representative subsystem in which all inequalities $\geq$ are replaced with equalities $=$. By discountedness, each such subsystem has a nondegenerate matrix, hence possesses a unique solution.

Proposition 2.3 *Let S be a full system of MD-constraints.*

1. *If x and y are feasible for S then z, defined by $z_i = \min(x_i, y_i)$ for $i \in \{1, \ldots, n\}$, is also feasible for S.*
2. *If S contains one constraint per group then the unique solution x^* of its unique representative equality subsystem is the least element of S and $x^* = \arg\min(1^T x | S)$.*
3. *$z = \arg\min(1^T x | S)$ is finite.*
4. *$z = \arg\min(1^T x | S)$ satisfies at least one constraint in each group as equality.*
5. *$z = \arg\min(1^T x | S)$ coincides with the unique solution to one of the representative equality subsystems of S.*

Proof.

1. Choose any constraint $a^T u \geq \beta$ from S. Suppose w.l.o.g. that $a_1 = 1$ and $z_1 = x_1$. Note that $a^T x = a^T \langle z_1, z_2 + \delta_2, \ldots, z_n + \delta_n \rangle \geq \beta$, where $z_i + \delta_i = x_i$ and $\delta_i \geq 0$ for $i \in \{2, \ldots, n\}$, implies $a^T z \geq \beta$, because the components a_i for $i \in \{2, \ldots, n\}$ are nonpositive. Indeed, $a^T x = a^T \langle z_1, z_2 + \delta_2, \ldots, z_n + \delta_n \rangle = a^T z + \sum_{i=2}^{n} a_i \delta_i \geq \beta$ implies $a^T z \geq \beta$, since the sum is nonpositive.

2. Let x be a feasible solution of $S \equiv Ax \geq b$ and let x^* be the unique solution of $Ax = b$. Then $A(x - x^*) \geq 0$ and $x - x^* \geq 0$ (or $x \geq x^*$) easily follows. Indeed, assuming $x \not\geq x^*$, select the smallest negative $x_i - x_i^*$ and get a contradiction with the discountedness of A.

3. Choose any representative equality subsystem S' of S, and let x^* be its unique finite solution. Since S' is a subsystem of S it follows that z is feasible for S', so by 2 we have $x^* \leq z$.

4. Otherwise, z would not be a minimum (if all inequalities in S_i were strict, then z_i could be decreased).

5. Follows from the above. □

Corollary 2.4 *For a full system S of MD-constraints there is a unique least element in the convex polyhedron defined by S, which coincides with the unique optimal solution to the linear program* $\min(p^T x|S)$*, where p is any positive vector.* □

We now sketch the two simple methods of computing least elements of full MD-systems, both based on Proposition 2.3. The first one is straightforward. It suffices to find a representative equality subsystem with the unique solution (found by Gaussian elimination) being a feasible solution to the whole system. This can be done by a straightforward exhaustive search in time $O((n^3 + m) \cdot \Pi_{i=1}^{n} n_i)$, where n_i is the number of inequalities for the i-th variable (the size of the i-th group).

A less straightforward method is to fix a representative equality subsystem S' and find its unique solution x'. If x' satisfies all other constraints of the system, it is the required least element. Otherwise, take *any* violated constraint and use it instead of the constraint in S' from the same group. The new unique solution x'' of the resulting representative equality subsystem is in the feasible region of S' (considered as inequalities $\geq$), of which x' is the least element. Therefore, $x' < x''$, which guarantees monotonicity and termination. However, the worst case bound remains the same. Below we will suggest more efficient methods for MD2-linear programs.

2.1 Two Types of MD-Systems

Due to the asymmetry of the discounting factors (negative components of a), we need to distinguish between two types of MD-systems: the $\geq$-type, defined above, and the $\leq$-type, which differs from the above by reversing the direction of all inequalities.

The full $\geq$-type MD-systems are appropriate for the problem of finding the least element of the feasible region, which coincides with the problem of finding the unique minimum of the function $1^T x$ on the feasible region.

Symmetrically, the full $\leq$-type MD-systems are appropriate for the problem of finding the *greatest* element of the feasible region, which coincides with the problem of finding the unique *maximum* of the function $1^T x$ on the feasible region.

These problems are however easily reducible to each other, since $Ax \leq b$ iff $A(-x) \geq -b$. Thus it is sufficient to give algorithms for solving the $\geq$-type problem, as we will do in the remainder of this paper.

We also have the following simple, but useful, connection.

Proposition 2.5 *If x is feasible for a full $\geq$-type MD-system and y is feasible for the reversed system, then $y \leq x$.*

Proof. Choose any representative equality subsystem, and let z^* be its unique solution. Then, by Proposition 2.3 and symmetry, we have $y \leq z^* \leq x$. □

3 Equal Discounting Factors

We first consider the case of MD2-LEP when the constraints have equal discounts, i.e., there is a single discounting factor λ, where $0 < \lambda < 1$, and all constraints are of the form $x_i \geq \lambda x_j + \beta_{ij}$ (if $i = j$ then this is equivalent to $x_i \geq \beta_{ii}/(1-\lambda)$). For this case, we may assume that there is at most one constraint per ordered pair of variables. In the following, we assume that x is a feasible solution to the full system of constraints.

Consider the weighted directed graph G with the variables of the constraint system as vertices, where a constraint $x_i \geq \lambda x_j + \beta_{ij}$ is represented by an edge from x_i to x_j with weight β_{ij}. An edge is called *tight* if the corresponding constraint is satisfied as equality. Tight edges may be colored red, but at most one outgoing red edge per vertex is allowed. A vertex with no outgoing red edge is called a *root*.

Proposition 3.1 *If every vertex has an outgoing red edge (i.e., there are no roots), then x is the unique minimal solution of the constraint system.*

Proof. Follows from Proposition 2.3. □

This suggests the following approach: starting from a feasible solution, we move in a nonpositive direction within the feasible region, making sure the number of red edges is nondecreasing. This has a clear intuitive interpretation as pulling a red tree down by its root, as explained below.

Definition 3.2 *A* red zone *is a maximal subgraph G' of G induced by red edges such that for every two vertices u, v of G' there is a vertex of G' reachable from both u and v by red paths (possibly empty) in G'. Every red zone is either a* red tree *with a root (as defined above), or a* red sun *with a unique cycle and incoming rays.*

Definition 3.3 *A* pull-down *of a red tree is performed as follows. The coordinates of x corresponding to nodes in the tree are decreased in such a way that the tree edges remain tight, while all other coordinates are kept fixed, until some constraint is just about to be violated. An edge corresponding to such a constraint (satisfied as equality) is then chosen, and is made the unique new outgoing red edge from its tail.*

Note that the tail of this new red edge will always be a node originating from the red tree that was pulled down, due to the monotonicity of the constraints. Also, roots are never created, i.e., once a vertex has acquired an outgoing red edge, it will always have *some* outgoing red edge. Thus, a pull-down results in one of the following three events, illustrated in Figure 1.

1. The root is *eliminated*, i.e., the whole tree grafts into a red zone (possibly itself, thereby creating a sun). This decreases the number of red trees by one.

2. A proper subtree *defects*, i.e., the new outgoing red edge of some non-root node connects the node and its predecessors to another red zone.

3. A non-root node makes an *internal switch*, i.e., reconnects within the same tree.

   Intuitively, this may happen for the following reason (formalized in Proposition 3.4). Every node x_i in a red tree is expressible as a linear function $x_i = \lambda^k x_j + \beta$ of its root

x_j with the slope λ^k. By pulling the root down (decreasing x_j), we also decrease x_i. At some point, a non-red edge between two nodes in the red tree may become tight and create an alternative path giving rise to the new function $x_i = \lambda^l x_j + \beta'$ with a smaller slope (the necessary condition for intersection), i.e., $l > k$, and an internal switch occurs.

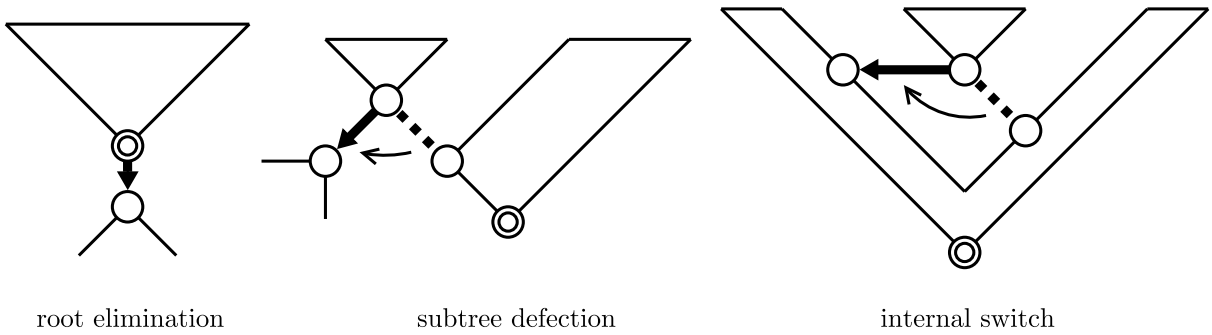

Figure 1: *The possible results of a pull-down operation.*

Algorithm 1, given in pseudo-code below, starts with a feasible solution (lines 2–3) and no red edges (line 4), and then repeatedly chooses a root and eliminates it by repeated pull-downs (lines 9–20), until no roots remain. By Proposition 3.1, this will yield the unique minimal solution, provided the algorithm terminates. We will now prove that it does terminate, and also derive an upper bound on its running time.

Proposition 3.4 *If an internal switch occurs, then the depth of the switching node increases.*

Proof. During a pull-down, the feasible solution x is moving in the direction $-s$ for some nonnegative vector s. We will call s_i the *speed* of x_i, and normalize s by making the speed of the root equal to 1. In order to keep a tree edge from x_i to x_j tight, we must have $s_i = \lambda s_j$, hence the speed of any tree node x_i must be $\lambda^{\text{depth of } x_i}$. For any vertex x_i not in the tree, we have $s_i = 0$.

Consider an edge from a tree node x_i to a tree node x_j, and suppose that the depth of x_i is strictly greater than the depth of x_j. Then $s_i \leq \lambda s_j$, and, since x is feasible, we have $x_i \geq \lambda x_j + \beta_{ij}$. This implies $(x_i - \tau s_i) \geq \lambda(x_j - \tau s_j) + \beta_{ij}$ for any non-negative τ, i.e., the constraint will never be violated.

Thus, before an edge from x_i to x_j is colored red due to an internal switch, the depth of x_i must be less or equal to the depth of x_j, hence the switch increases the depth of x_i. □

Corollary 3.5 *The number of successive pull-downs needed to eliminate the root of a red tree is at most t^2, where t is the number of nodes in the tree before the first pull-down.*

Proof. By Proposition 3.4, a node can perform at most $t - 1$ internal switches, and at most one defection, during the pull-downs. □

Algorithm 1: Solves MD2-LEP with equal discounting factors.
Input: An MD2-system represented by a weighted graph $G = (V, E, w)$ and a discounting factor $0 < \lambda < 1$. An edge $(x_i, x_j) \in E \subseteq V^2$ represents the linear constraint $x_i \geq \lambda x_j + w(x_i, x_j)$.
Output: The least element of the feasible region of the input system.

MD2$^{=}$-LEAST-ELEMENT(V, E, w, λ)
(1) ▷ Compute a feasible solution.
(2) $\xi \leftarrow \max_{e \in E} \dfrac{w(e)}{1-\lambda}$
(3) $value[V] \leftarrow \xi$
(4) $color[E] \leftarrow black$
(5) **while** there is a root r of a smallest red tree
(6) ▷ Eliminate r.
(7) **while** r is the root of a red tree T
(8) ▷ Pull down T.
(9) $V_T \leftarrow$ nodes of T
(10) $speed[V] \leftarrow 0$
(11) **foreach** $v \in V_T$ by pre-order tree traversal
(12) $speed[v] \leftarrow \lambda^{\text{depth of } v \text{ in } T}$
(13) $E_T \leftarrow$ black outgoing edges from V_T
(14) $time[E_T] \leftarrow \infty$
(15) **foreach** $(u, v) \in E_T$
(16) **if** $speed[u] > \lambda \cdot speed[v]$
(17) $time[(u, v)] \leftarrow \dfrac{\lambda \cdot value[v] - value[u] + w(u, v)}{\lambda \cdot speed[v] - speed[u]}$
(18) $(u, v) \leftarrow \arg\min_{e \in E_T} time[e]$
(19) **foreach** $x \in V_T$
(20) $value[x] \leftarrow value[x] - speed[x] \cdot time[(u, v)]$
(21) $color[$outgoing edges from $u] \leftarrow black$
(22) $color[(u, v)] \leftarrow red$
(23) **return** $value$

The bound in Corollary 3.5 is asymptotically tight — it is possible to give an example when $\Theta(t^2)$ pulls down are needed to eliminate one root, as will be done below.

Corollary 3.5 immediately yields an $O(n^3)$ upper bound on the total number of pull-downs needed to eliminate all roots. However, we will now prove that, by using a *greedy* strategy, always selecting the root of the smallest tree to be eliminated, just $O(n^2)$ pull-downs suffice.

Proposition 3.6 *By using the greedy strategy, all roots can be eliminated using $O(n^2)$ pull-downs.*

Proof. We always eliminate the root of the smallest tree. If there are k trees, at least one of them must have at most $\lfloor n/k \rfloor$ nodes, and thus the total number of pull-downs is

bounded above by

$$\sum_{k=1}^{n} \left(\frac{n}{k}\right)^2 \leq n^2 \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2 n^2}{6},$$

which is $O(n^2)$. □

A single pull-down operation is performed by the algorithm in $O(m)$ time as follows. First, the speeds (of decrease) for all vertices are computed (lines 10–12). Then, for each outgoing edge from a tree node, the algorithm determines whether the edge would ever be violated (line 16), and if so, calculates time until it this would happen (line 17). Finally, an edge with the minimal time until violation is chosen (line 18), the values of tree nodes are decreased, and edge colors are updated.

This results in the following

Theorem 3.7 *The running time of Algorithm 1 is $O(mn^2)$.*

Proof. By Proposition 3.6, since a single pull-down operation is performed in $O(m)$ time. □

A sample run of Algorithm 1 is given in Figure 2. It illustrates an important property: an edge may become red more than once. If this had not been the case, then an $O(m)$ bound on the total number of pull-downs would easily have followed.

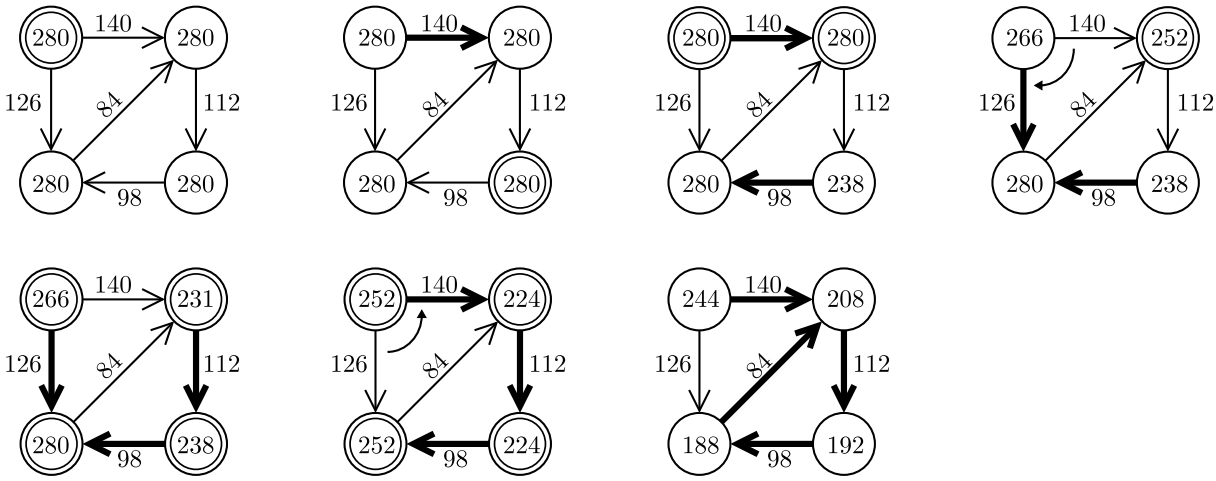

Figure 2: *Step by step illustration of the pull-downs performed by the algorithm on a particular input with $\lambda = 1/2$. Marked vertices are nodes of the red tree chosen for the next pull-down.*

The worst-case running time of Algorithm 1 is in fact $\Theta(mn^2)$, since it is possible to construct inputs of any size for which the algorithm proceeds as follows.

First, a single red tree spanning the entire graph is grown. It consists of a chain of k vertices $x_1, \ldots, x_k$ connected by red edges of weight 0 and another k vertices $y_1, \ldots, y_k$ connected by red edges with weights β_{j1} to the root vertex x_1. There are also black edges from every vertex y_j to every vertex x_i, $i \in \{2, \ldots, k\}$, with weights β_{ji}. The root x_1

has a black self-loop. We illustrate in Figure 3, for simplicity, the case $k = 3$, which straightforwardly generalizes to any k.

Pulling down the root x_1 causes the value of y_j to decrease according to $y_j = \beta_{j1} + \lambda x_1$. An internal switch occurs when the linear function $y_j = \beta_{j1} + \lambda x_1$ intersects $y_j = \beta_{ji} + \lambda^i x_1$ with $i > 1$, and y_j "jumps" up the chain, because the slope λ^i is less steep than λ. The algorithm terminates when the self-loop of x_1 becomes tight.

By choosing suitable weights, we can guarantee that the value of each y_j during the pull-downs will be a piecewise linear function of x_1 consisting of k pieces with slopes λ^l for decreasing $l = k, \ldots, 1$. This will cause all vertices y_j to climb the chain *one level at a time* and result in a total of $\Theta(k^2) = \Theta(n^2)$ internal switches.

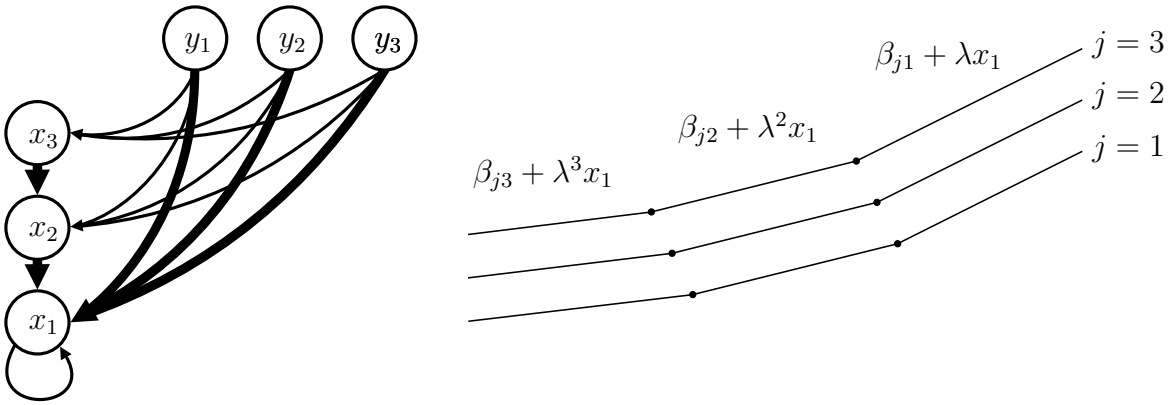

Figure 3: *A total of* 6 *internal switches can occur as the* y_j *climb the chain.*

4 Different Discounting Factors

In this section we address the case when constraints may have different discounting factors. Proposition 3.4 requires equal discounts, and although lifting this assumption still leaves us with a variant — the product of discounts on the path to the root decreases — and thereby a proof of termination, it spoils the strongly polynomial bounds. Therefore, we use a different method for this case, at the cost of slightly worse time complexity.

Hochbaum and Naor [7] suggested a simple and fast deterministic $O(mn^2 \log m)$ algorithm for finding a *feasible* solution of any linear system with at most two variables per inequality. We will show how to modify their algorithm so that it can be used to solve our optimization problem MD2-LEP.

The algorithm in [7] is based on the Fourier-Motzkin elimination method [9]. To eliminate a variable x_i, all inequalities containing x_i are replaced with inequalities $L \leq U$ for each pair $L \leq x_i$ and $x_i \leq U$ in the original system. Feasibility is obviously preserved, and the method can be applied recursively to compute a feasible solution, or determine that no one exists. However, the number of inequalities created during this process may be exponential.

In [7] the growth of the number of inequalities during the repeated Fourier-Motzkin elimination is controlled by removing a lot of inequalities, and adding a few, in a way that preserves feasibility. This is done by searching for feasible values between breakpoints (defined below).

For any two distinct variables x_i and x_j, the feasible region of the subsystem of inequalities containing only x_i and x_j lies between an upper and lower envelope, which are piecewise linear functions in the x_ix_j-plane. We denote the set of breakpoints of these functions by $B(x_i, x_j)$.

The original algorithm is focused on finding *any* feasible solution. Feasibility is trivial in our case, and the algorithm must be modified. We now state the modified version, and refer the reader to [7] for a detailed description of the original algorithm.

4.1 Algorithm 2

For each variable x_i, we denote by $x_i^{\min}$ the smallest value of x_i in any feasible solution to the original system of inequalities. By Proposition 2.3, $\langle x_1^{\min}, \ldots, x_n^{\min} \rangle$ is the unique minimal element of the feasible region. Using Proposition 2.5 and 2.2, we compute values a_i and b_i such that $a_i \leq x_i^{\min} \leq b_i$, and add the inequalities $a_i \leq x_i \leq b_i$ to the system.

We then perform steps **(i)**–**(iv)** for $i = 1, \ldots, n-1$ and maintain the following invariant: *before the i-th iteration,* $\langle x_i^{\min}, \ldots, x_n^{\min} \rangle$ *is a feasible solution to the current set of inequalities.*

(i) Let $B = \langle b_1, \ldots, b_k \rangle$ be the sorted sequence of x_i-coordinates of the breakpoints $\bigcup_{i<j\leq n} B(x_i, x_j)$.

To maintain the invariant, we must make sure that $x_i^{\min}$ remains a feasible value for x_i after step **(iii)**.

(ii) Using the procedure of Aspvall and Shiloach [1] (which, given any value α, decides if $\alpha < x_i^{\min}$), perform a binary search in B to find b_l and b_{l+1} such that $b_l \leq x_i^{\min} \leq b_{l+1}$ (if there is no b_l such that $b_l < x_i^{\min}$, we must have $x_i^{\min} = b_1$).

(iii) Add the inequalities $b_l \leq x_i$ and $x_i \leq b_{l+1}$, and discard any redundant inequalities.

For any x_j, there will now be at most two inequalities containing x_i and x_j.

(iv) Apply the Fourier-Motzkin elimination method to x_i.

Since Fourier-Motzkin elimination preserves feasible ranges for the remaining variables, the invariant is preserved.

After $n-1$ iterations of steps **(i)**–**(iv)**, what remains is x_n and two inequalities $\alpha \leq x_n$ and $x_n \leq \beta$, where α and β are constants. By the invariant, we have $\alpha = x_n^{\min}$, and thus we assign α to x_n. Backtracking, i.e., restoring previously discarded inequalities containing x_{n-1} and x_n, we assign to x_{n-1} the minimum feasible value with respect to these inequalities and the value assigned to x_n. By the invariant and monotonicity, continuing in this fashion for $x_{n-2}, \ldots, x_1$ gives us the optimal solution.

Our modifications do not significantly affect the worst case analysis in [7], and this results in

Theorem 4.1 *The running time of Algorithm 2 is* $O(mn^2 \log m)$. □

5 Applications

Solving the MD2-LEP has the following obvious optimal control interpretation. Starting in a state x_i, an agent selects one of a few available actions. Depending on the choice j, he gets paid some amount β and causes some inflation rate λ, and the next day everything repeats from the state x_j, ad infinitum. An optimal agent's strategy is described by the MD2-LEP instance with constraints $x_i \geq \beta + \lambda x_j$ defining the possible actions.

5.1 Two-Player Discounted Payoff Games

The model above is also known as a one-player discounted payoff game (DPG). The two-player version of a DPG also has a second player who, alternating the actions with the first one, tries to make the payoff as small as possible. Such games are known to be solvable in pure positional (memoryless) strategies for both players [10].

Algorithm 1 from Section 3, when combined with the randomized combinatorial optimization game techniques [2], provides for the best currently available algorithms for the two-player DPGs.

Theorem 5.1 *A two-player DPG can be solved in randomized subexponential time*

$$mn^2 \cdot \min\big(f(n_{\max}, m_{\max}),\ f(n_{\min}, m_{\min})\big), \tag{1}$$

where

$$f(n, m) = e^{2\sqrt{n \ln(m/\sqrt{n}) + O(\sqrt{n} + \ln m)}} \tag{2}$$

and n_π, m_π is the number of vertices and edges of player $\pi \in \{\min, \max\}$. □

Algorithm 2 gives a similar bound for the case of a generalized DPG, with different discounting factors. For previous algorithms, the bound was roughly the square of (1), more precisely, $f(n_{\max}, m_{\max}) \cdot f(n_{\min}, m_{\min})$.

5.2 Ergodic Partition for Mean Payoff Games

We also obtain a new *strongly subexponential* algorithm for solving the *ergodic partition* problem for mean payoff games (MPG): given an MPG find a partition of its vertices into subsets with equal values, together with their values [6].

Note that the algorithm from [3] is not strongly subexponential, it has a bound of the form $\log(W) \cdot 2^{O(\sqrt{n \log n})}$, where W is the maximum absolute edge weight. The latter algorithm proceeds by bisecting the range of possible values (hence the factor $\log W$), each time solving an instance of the longest shortest paths problem (in strongly subexponential time). Now we can give a strongly subexponential algorithm.

Theorem 5.2 *The MPG ergodic partition problem can be solved in randomized strongly subexponential time* (1). □

Proof. Reduce an MPG instance to a DPG instance, as in [11]. This reduction does not change the game graph, just adds an appropriately selected discounting factor. Finding values of this DPG can be done as explained in the previous section. The MPG values may be recovered from the DPG values. As a by-product, this algorithm also computes optimal strategies. □

6 Conclusions

Motivated by applications to one- and two-player games, we constructed two new strongly polynomial algorithms for finding unique optimal solutions to system of linear 2-variable monotonic inequalities, running in time $O(mn^2)$ and $O(mn^2 \log m)$ for equal and different discounting factors, respectively. Interestingly, there remains a big gap between finding feasible solutions to such systems in $O(m)$ time and solving them to optimality. Also our algorithm for different discounting factors does not improve[3] asymptotically over the feasibility algorithm [7] for arbitrary 2-variable constraints. Designing new, more efficient algorithms to narrow the existing gaps, or "proving" natural lower bounds represent challenging algorithmic problems. Note that our equal discounts algorithm, incidentally, has the same complexity as Bellman-Ford's shortest paths algorithm run from each vertex.

Extending the techniques to solve, in strongly polynomial time, MD-linear programs, with no restriction on the number of variables per inequality, is another challenging problem, with important consequences for linear programming, game theory, and linear complementarity.

References

[1] B. Aspvall and Y. Shiloach. Polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9:827–845, 1980.

[2] H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

[3] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 2006. Accepted for publication, to appear.

[4] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23(6):1313–1347, 1994.

[5] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementarity Problem.* Academic Press, 1992.

[6] V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

[7] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.

[8] K. G. Murty and F.-T. Yu. *Linear Complementarity, Linear and Nonlinear Programming.* Heldermann Verlag, Berlin, 1988. http://ioe.engin.umich.edu/people/fac/books/murty/linear_complementarity_webbook/.

[3] Actually, it does not exploit discountedness.

[9] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1986.

[10] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–110, 1953.

[11] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.