Improved Combinatorial Algorithms for Discounted Payoff Games

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science

from

Uppsala University
Department of Information Technology

by

Daniel Andersson

August 24, 2006

Advisor:	Dr. Sergei Vorobyov
Reviewer:	Prof. Wang Yi
Examiner:	Dr. Anders Jansson
Thesis Level:	20 credits (D-level)
Study Program:	Naturvetarprogrammet

Abstract

This thesis is devoted to the design and analysis of combinatorial algorithms for solving one-player versions of several prominent infinite duration games pertinent to automated verification of computerized systems.

We present the first two strongly polynomial algorithms for solving one-player discounted payoff games, running in time $\mathcal{O}(mn^2)$ and $\mathcal{O}(mn^2 \log m)$, where the latter algorithm allows edges to have different discounting factors. As applications, we are able to improve the best previously known strongly subexponential algorithms for solving two-player discounted payoff games and the ergodic partitioning problem for mean payoff games.

Keywords: discounted payoff game, strongly polynomial algorithm, combinatorial linear programming, mean payoff game, ergodic partition.

Preface

The purpose of this thesis is to document and summarize the research that I have conducted — in close collaboration with my advisor, Dr. Sergei Vorobyov — during the past six months at the Department of Information Technology of Uppsala University.

The thesis consists of a brief survey followed by two technical research papers. The survey aims at providing an accessible motivation and introduction to the research field and our contributions — leaving formal definitions, precise statements, and proofs to the technical papers. It also contains a short summary in Swedish.

In closing, I express my sincere gratitude to Sergei, who initiated me into computer science research and fostered my academic ambitions. Not only has his support been invaluable for my present work, but it has also given me inspiration and hope for the future.

Uppsala, August 2006 *Daniel Andersson*

Contents

1 Motivation

As we become more and more dependent upon computerized systems, the need to ensure their correctness increases dramatically. For a system whose failure or malfunction could cause major damage, it is desirable to have a formal proof of correctness, obtained by exhaustively examining every possible behavior of the system to verify that it has all the required properties. Because of the growing size and complexity of hardware and software systems, it is generally infeasible for the designers or programmers to perform such verification manually, and a wide range of automated verification techniques have been developed.

One appealing approach to the process of verification is to model it as a two-player game between a system and its environment. The system is then correct if it possesses a winning strategy against any malicious behavior of the environment. Taking this approach allows us to draw upon the rich mathematical theory of games — an established field of research, extensively developed during the 20^{th} century by prominent mathematicians such as John von Neumann (1903–1957), Lloyd Shapley (1923–), and John Nash (1928–) and now an indispensable tool in economic theory as well as various other disciplines, such as biology, philosophy, sociology, and political science. Game theory has also found numerous other applications in computer science, ranging from the characterization of complexity classes to the design and modeling of computer networks.

Unfortunately, much of the classical game theory is non-constructive, asserting the existence of various objects without providing any efficient methods for actually constructing them. For many applications, including verification, any practical usability of a game-theoretic approach requires the development of efficient algorithms for computational problems posed by games, such as determining the winner of a game or even finding optimal strategies for the players — usually called *solving* the game. Determining the complexity of such problems is considered to be one of the great challenges for theoretical computer science in the 21^{st} century [14].

This thesis focuses on that challenging problem for the family of games

including *parity games* [8], *mean payoff games* [6], *discounted payoff games* [18], and *simple stochastic games* (with arbitrary probability distributions) [16, 5]. All of them are two-player non-cooperative games played by moving a pebble around a finite graph for a possibly infinite duration. Their relevance for verification stems from the fact that they are closely connected to the *μ-calculus* — a modal fixpoint logic capable of expressing many interesting properties of systems, with an expressive power subsuming that of most other popular logics used for verification, including LTL, CTL, and CTL* [7]. Solving a game from any of the aforementioned classes corresponds to determining whether a system has a property expressible in the μ-calculus — a problem known as *μ-calculus model-checking*. The games are also interesting from a complexity-theoretic point of view, since they give rise to problems that are among the few natural ones known to belong to the complexity class $\mathbf{NP} \cap \mathbf{coNP}$ but not known to be in $\mathbf{P}$. Another, very well studied, problem with this property is integer factorization and, until recently, so was primality testing [15, 1].

At present, solving these games appears to be hard — the best known upper bound is randomized subexponential time [4]. The aim of this thesis is to investigate their complexity in the special case where a single player controls the game. Although these one-player games are simpler than their two-player counterparts, they still retain much of the original structure, and therefore, investigating them may provide valuable insights into the two-player games. Furthermore, solving a one-player game is equivalent to finding an optimal counterstrategy against a fixed positional strategy of the opponent, which is an underlying operation in the best known algorithms for the two-player games. Thus, more efficient algorithms for the one-player games automatically lead to improvements of their two-player counterparts.

One of the most important open problems in mathematical optimization is to determine the combinatorial complexity of *linear programming* — the optimization of a linear function subject to a system of linear constraints. The ellipsoid algorithm of Khachiyan [13] and the interior-point algorithm of Karmarkar [11] both have a running time polynomial in the total size of the input, but the number of arithmetic operations depends not only on the dimensions of the system of constraints, but also on the magnitude of the coefficients. The quest for a *strongly polynomial* algorithm, where the number of arithmetic operations is polynomial in the number of variables and constraints but independent of the coefficients, has continued for the last quarter of a century.

Particular cases of this fundamental problem appear when solving the one-player games. Whereas strongly polynomial algorithms for one-player parity and mean payoff games have been known for a long time [12], the

only known way to efficiently solve one-player discounted payoff or simple stochastic games has been to formulate them as linear programs and then apply general, non-strongly polynomial, linear programming algorithms. The aim of this thesis is to develop the first strongly polynomial algorithms for these games, by exploiting the special structure of the linear programs generated by them — something which has been successful for several other prominent problems, such as min-cost maximum flow [17].

2 Contributions of this Thesis

The primary contributions of this thesis are two new algorithms for solving one-player discounted payoff games [2, 3]. They are, to our knowledge, the first strongly polynomial algorithms for this problem, and they provide for improvements to the best known strongly subexponential algorithms for solving two-player discounted payoff games and the ergodic partition problem for mean payoff games [9, 18].

The first algorithm that we describe [2], is an adaptation of a previously existing algorithm for the special case of *linear feasibility* (finding a solution to a given set of linear constraints or determining that none exits) where each constraint contains at most two variables [10]. For the linear programs generated by one-player discounted payoff games, finding a feasible solution is trivial — the difficulty lies in finding the *optimal* solution. The fact that we can still use the same approach, depends on the special structure of our programs. One crucial property is that the constraints are *monotonic*, i.e., they have the form $x_i \leq \alpha x_j + \beta$ with $\alpha > 0$. Our modifications do not affect the asymptotic running time: $\mathcal{O}(mn^2 \log m)$, where n is the number of vertices/variables and m is the number of edges/constraints. The resulting algorithm is general, in the sense that it works equally well for a natural generalization of the standard discounted payoff games, in which edges have individual, possibly different, discounting factors instead of a common global one.

Our second algorithm [3], uses a novel approach. Starting from an initial feasible point, the algorithm proceeds by descending along the boundary of the feasible region, until the optimum is reached, and the descent is guided by a rule stating that the number of constraints satisfied as equality should be non-decreasing. Considering the trees formed by the edges corresponding to such constraints, and studying their evolution during the descent, we are able to prove tight asymptotic bounds on the number of elementary operations performed by the algorithm. Its total running time is $\mathcal{O}(mn^2)$. This is a slight improvement over the previously described algorithm, but the bound only applies to the standard discounted payoff games.

We have also been investigating one-player simple stochastic games, and we have devised several methods for solving them. These methods were the inspiration for our descent algorithm for the discounted payoff games, and we used an implementation of this algorithm to find an explicit counterexample to a promising conjecture for the more general methods [3]. Proving strongly polynomial bounds for any of our methods remains a challenging open problem for future research.

3 Dissemination of Results

In addition to the technical research papers [2, 3] found in the appendix, the results contained in this thesis have also been presented as follows.

- Poster presentation "Improved algorithms for discounted payoff games" at the *BRICS Summer School on Game Theory in Computer Science* in Aarhus, Denmark, June 2006.
- Oral presentation "Are one-player games always simple?" at the Isaac Newton Institute workshop *Games and Verification* — the 2006 Annual Meeting of the EU Research Training Network *Games and Automata for Synthesis and Validation* — in Cambridge, UK, July 2006.
- Oral presentation "An improved algorithm for discounted payoff games" at the *18^{th} European Summer School in Logic, Language and Information* in Málaga, Spain, August 2006.

4 Future Research

Since our strongly polynomial bounds for solving one-player discounted payoff games are the first ones known, it is not unlikely that they can be further improved. For instance, our adapted feasibility algorithm does not fully exploit the special structure of the linear programs generated by the games.

An interesting observation regarding such programs is that there currently appears to be a big gap in complexity between finding feasible solutions and finding optimal solutions. Investigating whether this gap is inherent, by designing more efficient algorithms or establishing natural lower bounds, is a challenging problem for future research.

However, the most intriguing question for further research is whether our results can be extended to one-player simple stochastic games. We have developed several candidate algorithms, and although they are likely to be efficient in practice, bounds on their worst case behavior are elusive. Drawing inspiration from advanced optimization techniques, such as interior point methods, we continue our efforts to solve this important open problem.

5 Summary in Swedish

Allteftersom vi blir mer och mer beroende av datoriserade system, ökar vårt behov av att kunna försäkra oss om att de fungerar som avsett. I de fall där ett systemfel skulle kunna orsaka stor skada är det önskvärt att göra en uttömmande undersökning av alla möjliga beteenden för att verifiera att systemet besitter alla nödvändiga egenskaper. På grund av den växande storleken och komplexiteten hos hårdvaru- och mjukvarusystem är det i allmänhet omöjligt för konstruktörerna att genomföra sådan verifiering manuellt, och en mängd olika metoder för automatisk verifiering har därför utvecklats.

Ett sätt att se verifieringen av ett system är att betrakta situationen som ett spel mellan systemet och dess omgivning. Systemet är i så fall korrekt om det besitter en vinnande strategi mot den illasinnade omgivningen. Att anamma detta synsätt låter oss dra nytta av den rika matematiska spelteorin — ett etablerat forskningsområde, utvecklat under 1900-talet av många framstående matematiker och idag ett ofta använt verktyg inom såväl ekonomi som biologi, sociologi och statsvetenskap. Spelteori har också funnit många andra tillämpningar inom datavetenskapen, från beskrivning av komplexitetsklasser till utformning och modellering av datornätverk.

Tyvärr så är en stor del av den klassiska spelteorin icke-konstruktiv, i betydelsen att den bevisar existensen av olika objekt utan att tillhandahålla någon effektiv metod för att faktiskt konstruera dem. För många tillämpningar, inklusive verifiering, är en förutsättning för att ett spelteoretiskt synsätt ska vara praktiskt användbart utvecklingen av effektiva algoritmer för spelteoretiska problem, såsom att beräkna optimala strategier för spelarna — vanligtvis kallat att *lösa* spelet. Att utveckla och analysera sådana algoritmer anses vara en av de stora utmaningarna för teoretisk datavetenskap i det nya århundradet [14].

Denna avhandling fokuserar på detta problem för den familj av spel som innehåller *paritetsspel* [8], *medelavkastningsspel* [6], *diskonterade spel* [18] och *enkla stokastiska spel* (med godtyckliga sannolikhetsfördelningar) [16, 5]. I samtliga så spelar två spelare mot varandra genom att växelvis flytta en markör runt en graf. Att de är relevanta för verifiering beror på att de är nära

sammankopplade med så kallad μ-kalkyl — en logik som kan användas för att uttrycka många intressanta egenskaper hos system, med en uttrycksförmåga som överstiger den hos de flesta andra logiker som används vid verifiering. Att lösa ett spel från någon av de tidigare nämnda klasserna motsvarar att avgöra huruvida ett system uppfyller ett krav uttryckt i μ-kalkyl.

Än så länge vet ingen hur, eller om, dessa spel kan lösas effektivt, även om betydande framsteg har gjorts på senare tid [4]. Syftet med denna avhandling är att undersöka svårigheten att lösa spelen i det specialfall där en ensam spelare styr spelet. Även om dessa *ensam-spel* är enklare än sina fulltaliga motsvarigheter, så bibehåller de mycket av den ursprungliga strukturen, och därför kan studiet av dem bidra till förståelsen för hur de fungerar med två spelare. Dessutom är problemet att lösa ett ensam-spel ekvivalent med att bestämma en optimal motstrategi mot en på förhand känd strategi hos motståndaren, en ofta använd operation i de bästa algoritmerna för två spelare. Förbättrade algoritmer för en spelare resulterar således i förbättrade algoritmer för två spelare.

I denna avhandling presenterar vi två nya algoritmer för diskonterade ensam-spel [2, 3]. De är de första kända algoritmerna för detta problem som är *starkt polynomiella* — antalet aritmetiska operationer de utför begränsas av ett polynom i antalet hörn och kanter i spelets graf. Hittills har det enda kända sättet att effektivt lösa spelen varit att använda allmänna, svagt polynomiella metoder för så kallad *linjär programmering* — optimering av en linjär funktion under linjära bivillkor. Detta gäller fortfarande för enkla stokastiska ensam-spel, som kan betraktas som en generalisering av de diskonterade. Att undersöka huruvida våra resultat kan utökas till denna större klass av spel är en utmanande uppgift för fortsatt forskning.

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math.*, 160(2):781–793, 2004.

[2] D. Andersson. An improved algorithm for discounted payoff games. In J. Huitink and S. Katrenko, editors, *Proceedings of the Eleventh ESSLLI Student Session*, pages 91–98, Málaga, Spain, June 2006.

[3] D. Andersson and S. Vorobyov. Fast algorithms for monotonic discounted linear programs with two variables per inequality. Manuscript submitted to *Theoretical Computer Science*, July 2006. Preliminary version available as Isaac Newton Institute Preprint NI06019-LAA.

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

[5] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[6] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journ. of Game Theory*, 8:109–113, 1979.

[7] E. A. Emerson. Model checking and the Mu-calculus. In N. Immerman and Ph. G. Kolaitis, editors, *DIMACS Series in Discrete Mathematics*, volume 31, pages 185–214, 1997.

[8] E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Annual IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[9] 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.

[10] 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.

[11] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[12] R. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[13] L. G. Khachiyan. A polynomial algorithm in linear programming. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

[14] C. Papadimitriou. Algorithms, games, and the internet. In *ACM Annual Symposium on Theory of Computing*, pages 749–753. ACM, July 2001.

[15] V. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4:214–220, 1975.

[16] L. S. Shapley. Stochastic games. *Proc. Natl. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.

[17] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.

[18] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Appendix — Technical Research Papers

This appendix includes the papers listed below.

- D. Andersson. An improved algorithm for discounted payoff games. In J. Huitink and S. Katrenko, editors, *Proceedings of the Eleventh ESSLLI Student Session*, pages 91–98, Málaga, Spain, June 2006.
- D. Andersson and S. Vorobyov. Fast algorithms for monotonic discounted linear programs with two variables per inequality. Manuscript submitted to *Theoretical Computer Science*, July 2006. Preliminary version available as Isaac Newton Institute Preprint NI06019-LAA.

An Improved Algorithm for Discounted Payoff Games

Daniel Andersson
Uppsala University
daniel@math.uu.se

Abstract. We show that an optimal counterstrategy against a fixed positional strategy in a generalized discounted payoff game, where edges have individual discounts, can be computed in $O(mn^2 \log m)$ strongly polynomial time, where n and m are the number of vertices and edges in the game graph. This results in the best known strongly subexponential time bound for solving two-player generalized discounted payoff games.

1 Introduction

Parity games, mean payoff games, and discounted payoff games constitute a chain of increasingly complex infinite two-player perfect information non-cooperative games played on finite graphs. They are closely related to μ-calculus model checking, popular in computer-aided program verification, and their associated decision problems all share the rare property of being interesting problems in the complexity class $\mathbf{NP} \cap \mathbf{coNP}$ with widely conjectured, but yet unproved, $\mathbf{P}$-membership.

In this paper we investigate the problem of finding an optimal counterstrategy against a fixed positional strategy in a generalized discounted payoff game (DPG), where edges have individual discounting factors. We present a strongly polynomial algorithm which, when used as a component in the randomized combinatorial optimization schemes of (Björklund and Vorobyov 2005), gives the best strongly subexponential algorithm for two-player DPGs currently known. The improvement from the previously best known one is roughly $\sqrt{T(n,m)}$ compared with $T(n,m)$.

The problem of solving DPGs is related to the following well-known combinatorial optimization problems.

Stopping simple stochastic games (Condon 1992). This is a more general class of two-player games allowing random choice vertices with probability distributions on their outgoing edges. Their associated decision problem also belongs to $\mathbf{NP} \cap \mathbf{coNP}$, and even the existence of a strongly polynomial algorithm for the one-player version is an open problem.

Proceedings of the Eleventh ESSLLI Student Session
Janneke Huitink & Sophia Katrenko (editors)

Generalized network flow problems (Cohen and Megiddo 1991). It turns out that the linear programs generated by one-player DPGs are dual to those generated by a particular case of the generalized transshipment problem.

The linear complementarity problem (Murty and Yu 1988; Cottle, Pang, and Stone 1992). Given a square matrix M and a vector q, find some vector $z \geq 0$ such that $Mz + q \geq 0$ and $z^T(Mz + q) = 0$. If M is a Z- and P-matrix, this can be done by solving a linear program whose feasible region contains a unique minimal element, and there is a strongly polynomial algorithm due to Chandrasekaran; see (Cottle et al. 1992). When the problem is generalized to allow several constraints for each variable (Cottle and Dantzig 1970), the minimal element property is preserved, but the existence of a strongly polynomial algorithm is an open problem.

2 Preliminaries

A *generalized discounted payoff game* (DPG) is a 5-tuple $(V_{\text{MAX}}, V_{\text{MIN}}, E, w, \lambda)$, where:

- V_{MAX} and V_{MIN} are disjoint sets of vertices belonging to the players MAX and MIN, respectively; $V = \{v_1, \ldots, v_n\}$ denotes $V_{\text{MAX}} \cup V_{\text{MIN}}$;
- $E = \{e_1, \ldots, e_m\}$ is a set of directed edges between vertices in V, such that each vertex has at least one outgoing edge; we allow multiple edges between the same ordered pair of vertices and denote the set of edges from u to v by $E(u, v)$; we denote the set of outgoing edges from v by $E(v)$; we define $E_p = \bigcup_{v \in V_p} E(v)$ for $p \in \{\text{MAX}, \text{MIN}\}$;
- $w : E \to \mathbb{Q}$ is a weight function;
- $\lambda : E \to \{x \in \mathbb{Q} : 0 < x < 1\}$ is a discount function.

The game is played as follows. First, a token is placed at some initial vertex. Then, the following step is repeated ad infinitum: the owner of the vertex where the token is currently placed chooses an outgoing edge from this vertex and then moves the token to the head of the chosen edge.

This results in an infinite *play* $\pi = e_{i_0} e_{i_1} \ldots$, and the objective of MAX and MIN is to maximize and minimize, respectively, its *value* $\mu(\pi)$, defined by

$$\mu(\pi) = w(e_{i_0}) + \lambda(e_{i_0})\big(w(e_{i_1}) + \lambda(e_{i_1})(\cdots)\big) = \sum_{j=0}^{\infty} \left(w(e_{i_j}) \prod_{0 \leq k < j} \lambda(e_{i_k}) \right). \tag{1.1}$$

A *pure positional strategy* σ for player $p \in \{\text{MAX}, \text{MIN}\}$ is a selection of exactly one outgoing edge from each vertex owned by p, i.e., an element of the set $\prod_{v \in V_p} E(v)$ which we denote by $\mathcal{P}_p$. A play where p only uses edges in σ is said to be *consistent* with σ.

It follows from (Shapley 1953; Zwick and Paterson 1996) that there exist $\nu : V \to \mathbb{Q}$ and pure positional *optimal strategies* for MAX and MIN ensuring $\mu(\pi) \geq \nu(v)$ and $\mu(\pi) \leq \nu(v)$, respectively, for any play π starting from v that is consistent with the respective strategy. Henceforth, all strategies will be assumed to be pure positional, unless otherwise stated.

To *solve* a DPG is to compute the values $\nu(v)$ for all $v \in V$. From these values an optimal strategy for any player $p \in \{\text{MAX}, \text{MIN}\}$ can be constructed by, for each vertex $u \in V_p$, selecting an edge $e \in E(u)$ such that $\nu(u) = w(e) + \lambda(e)\nu(v)$, where v is the head of e. Conversely, given an optimal strategy for each player, we can easily compute $\nu(v)$ for any $v \in V$ by noting that $\nu(v) = \mu(\pi)$, where π is the unique play starting from v that is consistent with the given strategies.

A *one-player* DPG is a DPG with $V = V_{\text{MAX}}$ or $V = V_{\text{MIN}}$. Given a strategy for one of the players, a corresponding *optimal counterstrategy* is an optimal strategy for the opponent in the one-player game obtained by removing all edges not used by the given strategy and assigning all vertices to the opponent. Below, we will present an algorithm for the problem of finding an optimal counterstrategy, and then show how it can be used as a component in an algorithm for solving general (two-player) DPGs.

In any DPG, the roles of MAX and MIN can be interchanged by, before and after the game is solved, changing the sign of each edge weight and computed vertex value, respectively. Thus, it suffices to consider the problem of finding an optimal counterstrategy for MIN against a given strategy for MAX, i.e., solving a one-player DPG with $V = V_{\text{MIN}}$, as will be done below.

3 Solving One-Player DPGs

3.1 Linear Programming Formulation

We consider the problem of solving a one-player DPG with $V = V_{\text{MIN}}$, i.e., for each vertex $v \in V$ finding the minimum weight $\nu(v)$ of any infinite "discounted path" from v. The vector $\langle \nu(v_1), \ldots, \nu(v_n) \rangle$ must be a feasible solution to the following system of inequalities:

$$x_i \leq w(e) + \lambda(e)x_j \quad \text{for all } v_i, v_j \in V \text{ and } e \in E(v_i, v_j). \tag{1.2}$$

This system has many special properties. We can easily find *some* feasible solution in $O(m)$ time, by noting that $\langle \xi, \ldots, \xi \rangle$ is feasible iff $\xi \leq \frac{w(e)}{1-\lambda(e)}$ for all $e \in E$. Furthermore, since $a_i \leq w(e) + \lambda(e)a_j$ and $b_i \leq w(e) + \lambda(e)b_j$ implies

$$\max\{a_i, b_i\} \leq \max\{w(e) + \lambda(e)a_j,\ w(e) + \lambda(e)b_j\} = w(e) + \lambda(e)\max\{a_j, b_j\}, \tag{1.3}$$

and similarly for the minima, the set of feasible solutions equipped with the binary operations of componentwise maximum and minimum forms a lattice. In particular, there can be at most one *maximal solution* x^* such that $x^* \geq a$ (i.e., $x_i^* \geq a_i$ for $i = 1, \ldots, n$) for any feasible solution a. To see that $\langle \nu(v_1), \ldots, \nu(v_n) \rangle$ is in fact the unique maximal solution, consider a play $e_{k_0}e_{k_1}, \ldots$ from $v_i \in V$ consistent with an optimal strategy for MIN and note that the corresponding chain of inequalities implies $x_i \leq \nu(v_i)$.

Thus, the problem can be stated as the following linear program:

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i=1}^{n} x_i \\ \text{subject to} & x_i \leq w(e) + \lambda(e) x_j \quad \text{for all } v_i, v_j \in V \text{ and } e \in E(v_i, v_j). \end{array} \tag{1.4}$$

3.2 General Feasibility Algorithms

Megiddo (1983) gave an $O(mn^3 \log m)$ strongly polynomial algorithm for finding a feasible solution to any linear program with n variables, m inequalities, and at most two variables per inequality. Since the algorithm also computes the feasible range for each variable, it can be used to solve our optimization problem (1.4). We could also use the deterministic $O(mn^2(\log m + \log^2 n))$ algorithm or the expected $O(n^3 \log n + mn \log^3 m \log n + mn \log^5 n)$ randomized algorithm of Cohen and Megiddo (1991).

Hochbaum and Naor (1994) suggested a simpler and faster deterministic $O(mn^2 \log m)$ algorithm for finding a feasible solution. However, their algorithm does not compute the feasible ranges explicitly. We will show how to modify it so that it can be used to solve our optimization problem (1.4).

The approach of (Hochbaum and Naor 1994) is to use the Fourier-Motzkin elimination method (Schrijver 1986). 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 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 a straightforward application of such repeated elimination may be exponential.

The algorithm in (Hochbaum and Naor 1994) limits the growth of the number of inequalities during the repeated Fourier-Motzkin elimination by simplifying the system before each elimination. Using a decision procedure by Aspvall and Shiloach (1980), the algorithm attempts to locate a small interval containing a feasible value for the variable to be eliminated. When the variable is restricted to this interval, all but $O(n)$ of the inequalities containing it can be identified as redundant, provided that the interval is sufficiently small.

In order to find such an interval in strongly polynomial time, the search is confined to certain "interesting" values. For any two distinct variables x_i and x_j, the feasible region of the subsystem of inequalities not containing variables other than x_i and x_j lies between an upper and lower envelope, which are piecewise linear functions in the $x_i x_j$-plane; we denote the set of breakpoints of these functions by $B(x_i, x_j)$. The interesting values will be projections of such breakpoints.

The original algorithm is focused on finding *any* feasible solution. Feasibility is trivial for our system (1.2), and the algorithm must be modified. We now state the modified version, which solves our optimization problem (1.4), and refer the reader to (Hochbaum and Naor 1994) for a detailed description of the original algorithm.

3.3 Modified Algorithm

First, for each variable x_i, we compute values l_i and u_i such that $l_i \leq x_i^* = \nu(v_i) \leq u_i$. As l_i we may take the i-th component of some feasible solution, which, as noted above, can be computed in $O(m)$ time. As u_i we may use the value $\mu(\pi)$ of any play π starting from v_i. We add to the original system (1.2) the inequalities $l_i \leq x_i$ and $x_i \leq u_i$ for $i = 1, \ldots, n$.

Then we perform the steps 1–5 below for $i = 1, \ldots, n-1$ and maintain the following invariant:

before the i-th iteration, $\langle x_i^*, \ldots, x_n^* \rangle$ is a feasible solution to the current system.

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

To maintain the invariant, we must make sure that x_i^* remains a feasible value for x_i after step 3.

2. Using the procedure of Aspvall and Shiloach (1980) (which, given any value ξ, decides whether $x_i^* < \xi$ in $O(mn)$ time), perform a binary search in B to find b_l and b_{l+1} such that $b_l \leq x_i^* \leq b_{l+1}$ (if there is no b_{l+1} such that $x_i^* < b_{l+1}$, then $x_i^* = b_k$).

3. Add the inequalities $b_l \leq x_i$ and $x_i \leq b_{l+1}$ to the current system.

4. For $j = i, \ldots, n$, discard all inequalities that are redundant with respect to all other inequalities that do not contain variables other than x_i and x_j.

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

5. 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 1–5, 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 $\beta = x_n^*$, and thus we assign β to x_n. Backtracking, i.e., restoring previously discarded inequalities containing x_{n-1} and x_n, we assign to x_{n-1} the maximum feasible value with respect to these inequalities and the value assigned to x_n. By the invariant and the lattice structure of the feasible region, 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 (Hochbaum and Naor 1994), and thus the running time is $O(mn^2 \log m)$.

3.4 Equal Discounts

In (Andersson and Vorobyov 2006), a different approach to the problem is presented. For the particular case when all edges have the same discount, the resulting algorithm has a running time of $O(mn^2)$, which is a slightly better bound than the above.

4 Solving Two-Player DPGs

We now consider the problem of solving a two-player DPG. Björklund and Vorobyov (2005) give a general scheme for optimizing a wide class of functions, which contains strategy evaluation functions for many infinite games — the so-called recursively local-global (RLG) functions. We present the scheme applied to the case of DPGs, and refer the reader to (Björklund and Vorobyov 2005) for a detailed description of the more general approach.

Let $\mathcal{P}_{\text{MAX}}$ be the set of all (pure positional) strategies for MAX in $(V_{\text{MAX}}, V_{\text{MIN}}, E, w, \lambda)$. A *face* $\mathcal{P}'_{\text{MAX}}$ of $\mathcal{P}_{\text{MAX}}$ is the set of strategies for MAX in a game $(V_{\text{MAX}}, V_{\text{MIN}}, E', w, \lambda)$ with $E' \subseteq E$. Furthermore, if E' is obtained from E by removing all but one of the outgoing edges from some vertex $v \in V_{\text{MAX}}$, then $\mathcal{P}'_{\text{MAX}}$ is called a *facet* of $\mathcal{P}_{\text{MAX}}$. Any two strategies that differ only for a single vertex are called *neighbors*.

Suppose that $\mathit{eval} : \mathcal{P}_{\text{MAX}} \to \mathbb{Q}$ computes the optimal value of the linear program (1.4) resulting from fixing MAX's strategy. To maximize *eval* on a face $\mathcal{P}'_{\text{MAX}} \subseteq \mathcal{P}_{\text{MAX}}$, starting from some $\sigma \in \mathcal{P}'_{\text{MAX}}$, we use the following randomized iterated improvement algorithm from (Björklund and Vorobyov 2005).

1. If $|\mathcal{P}'_{\text{MAX}}| = 1$, then return σ.
2. Otherwise, select uniformly at random a facet $\mathcal{F}$ of $\mathcal{P}'_{\text{MAX}}$ such that $\sigma \notin \mathcal{F}$.
3. Recursively find the maximum element σ^* of $\mathcal{P}'_{\text{MAX}} \setminus \mathcal{F}$ starting from σ.
4. Let σ' be the unique neighbor of σ^* on $\mathcal{F}$.
5. If $\mathit{eval}(\sigma') \leq \mathit{eval}(\sigma^*)$, then return σ^*.
6. Otherwise, recursively find and return the maximum element of $\mathcal{F}$ starting from σ'.

The correctness follows from the results on simple stochastic games in (Björklund and Vorobyov 2005), which also apply to DPGs. From the analyses done in (Kalai 1992; Matoušek et al. 1996) (described in (Björklund and Vorobyov 2005)), it follows that the expected number of calls to the subroutine for *eval* is $f(|V_{\text{MAX}}|, |E_{\text{MAX}}|)$, where

$$f(n, m) = e^{2\sqrt{n \ln(m/\sqrt{n}) + O(\sqrt{n} + \ln m)}}. \tag{1.5}$$

Using the strongly polynomial algorithm presented in the previous section to compute *eval*, and recalling that the roles of MAX and MIN can be interchanged by simple transformations, we thus get a strongly subexponential algorithm with an expected total running time of

$$\min\{f(|V_{\text{MAX}}|, |E_{\text{MAX}}|), f(|V_{\text{MIN}}|, |E_{\text{MIN}}|)\} \cdot mn^2 \log m. \tag{1.6}$$

This is an improvement compared to previously available algorithms for *eval*, which either resorted to non-strongly polynomial LP-solvers, or to once again applying subexponential iterated improvement algorithms similar to the one above, resulting in a total expected running time of, roughly, $f(|V_{\text{MAX}}|, |E_{\text{MAX}}|) \cdot f(|V_{\text{MIN}}|, |E_{\text{MIN}}|)$.

5 Conclusions

We have described a new, and currently the best available, strongly polynomial algorithm for solving one-player generalized discounted payoff games, and shown how it can be incorporated into an algorithm for the two-player version, reducing the running time to roughly $\sqrt{T(n,m)}$ from $T(n,m)$.

It is likely that these results can be further improved, by exploiting more of the special properties of the linear programs arising from DPGs. The natural next step is to investigate more general classes of one-player games, such as one-player simple stochastic games with arbitrary probability distributions, for which the existence of a strongly polynomial algorithm is still an open problem.

Acknowledgements

The author is grateful to Sergei Vorobyov for many useful comments and ideas.

References

Andersson, D. and S. Vorobyov (2006, May). Fast algorithms for monotonic discounted linear programs with two variables per inequality. Preprint NI06019-LAA, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK.

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

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

Cohen, E. and N. Megiddo (1991). Improved algorithms for linear inequalities with two variables per inequality. In *ACM Annual Symposium on Theory of Computing*, pp. 145–154.

Condon, A. (1992). The complexity of stochastic games. *Information and Computation 96*, 203–224.

Cottle, R. W. and G. B. Dantzig (1970). A generalization of the linear complementarity problem. *Journal of Combinatorial Theory 8*, 79–90.

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

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

Kalai, G. (1992). A subexponential randomized simplex algorithm. In *24th ACM STOC*, pp. 475–482.

Matoušek, J., M. Sharir, and M. Welzl (1996). A subexponential bound for linear programming. *Algorithmica 16*, 498–516.

Megiddo, N. (1983). Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput. 12*(2), 347–353.

Murty, K. G. and F.-T. Yu (1988). *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag. http://ioe.engin.umich.edu/people/fac/books/.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley and Sons.

Shapley, L. S. (1953). Stochastic games. *Proc. Nat. Acad. Sci. USA 39*, 1095–110.

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

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

Daniel Andersson [1] and Sergei Vorobyov [*,2,3]

Department of Information Technology,
Uppsala University, Box 337,
SE-751 05 Uppsala, Sweden

Abstract

We suggest two 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.

Key words: linear programming, strongly polynomial algorithm, mean payoff and discounted payoff games.

* Corresponding author. Phone: +46 18 471 10 55. Fax: +46 18 55 02 25.
Email address: Sergei.Vorobyov@it.uu.se (Sergei Vorobyov).

[1] Supported by the Emma and Anders Zorn Foundation.
[2] Support by the EPSRC Grant 531174 and by the VR Grant 50861101 is gratefully acknowledged.
[3] Participant of the Logic and Algorithms Programme organized at the Isaac Newton Institute, Cambridge, UK.

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.

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* [4] 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. [5]

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

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

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

one constraint 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 synchronously 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 used, as a subroutine, with the randomized techniques for two player games [2], 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 [3] and 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 $x^T(Ax + b) = 0$; see [8,5]. Recall that a square real matrix is called: 1) a Z-matrix if all of its off-diagonal elements are nonpositive; 2) a P-matrix if all of its 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 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 (i.e., at most 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 of 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 from each group of S. 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 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 previously, 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 $\lambda \in (0, 1)$, 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, x denotes 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 by x. 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 natural 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 of the tree 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.

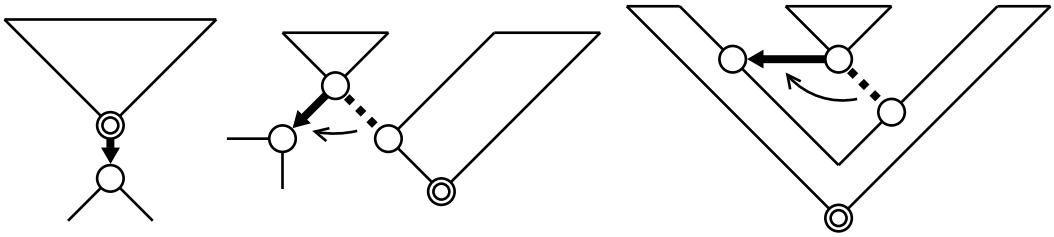

Figure 1. *The possible results of a pull-down operation.*

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.[6] 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

[6] Note that during a pull-down the root decreases the fastest.

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. □

The bound in Corollary 3.5 is asymptotically tight — below we give an example when $\Theta(t^2)$ pulls down are needed to eliminate one root.

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. □

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

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.

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.

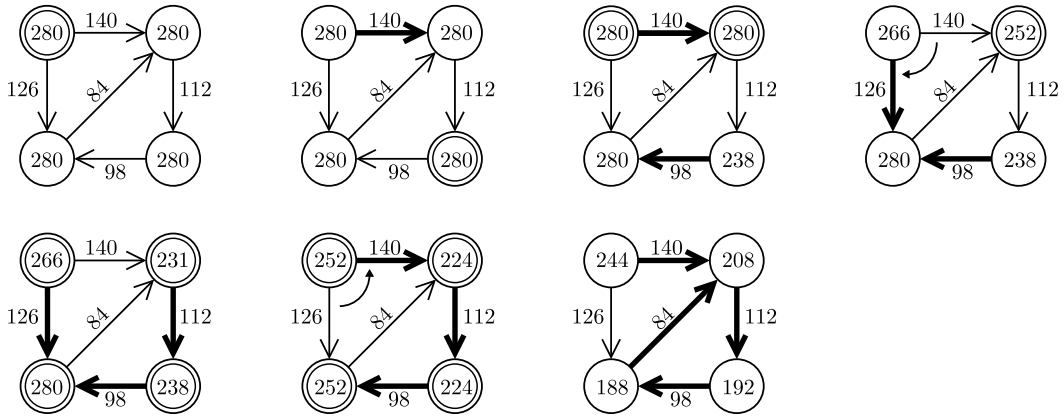

Figure 2. *Step by step illustration of the pull-downs performed by the algorithm on a particular input with* $\lambda = 1/2$. *Marked vertices are nodes of the red tree chosen for the next pull-down.*

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.

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 with every internal switch — 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 a slightly worse time complexity.

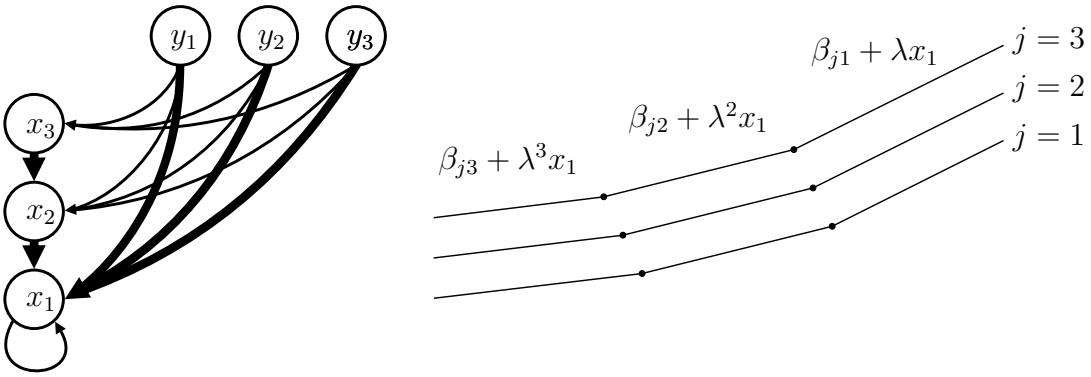

Figure 3. *A total of* 6 *internal switches can occur as the* y_j *climb the chain.*

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 approach of [7] consists in using 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 preserved, and the method can be applied recursively to compute a feasible solution, or determine that none exists. However, the number of inequalities created during a straightforward application of such repeated elimination may be *exponential*.

The algorithm in [7] limits the growth of the number of inequalities during the repeated Fourier-Motzkin elimination by simplifying the system before each elimination. Using a decision procedure by Aspvall and Shiloach [1], the algorithm attempts to locate a small interval containing a feasible value for the variable to be eliminated. When the variable is restricted to this interval, all but $O(n)$ of the inequalities containing it can be identified as redundant.

In order to find such an interval in strongly polynomial time, the search is confined to certain "interesting" values. For any two distinct variables x_i and x_j, the feasible region of the subsystem of inequalities not containing variables other than x_i and x_j lies between an upper and lower envelope, which are piecewise linear functions in the $x_i x_j$-plane; we denote the set of breakpoints of these functions by $B(x_i, x_j)$. The interesting values will be projections of such breakpoints on the x_i-axis.

The original algorithm [7] is focused on finding *any* feasible solution. Feasibility is trivial for our systems, and the algorithm must be modified. We now state the modified version, which solves our optimization problem MD2-LEP, 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^* the minimal value of x_i in any feasible solution to the original system of inequalities. By Proposition 2.3, $\langle x_1^*, \ldots, x_n^* \rangle$ 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 $a_i \leq x_i^* \leq b_i$, and add the inequalities $a_i \leq x_i \leq b_i$ to the system.

We then perform the steps **(i)**–**(v)** below for $i = 1, \ldots, n-1$ and maintain the following invariant:

> *before the i-th iteration,* $\langle x_i^*, \ldots, x_n^* \rangle$ *is a feasible solution to the current system of inequalities.*

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

To maintain the invariant, we must make sure that x_i^* remains a feasible value for x_i after the variable is restricted in step **(iii)**.

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

(iii) Add the inequalities $b_l \leq x_i$ and $x_i \leq b_{l+1}$ to the current system.

(iv) For $j = i, \ldots, n$, discard all inequalities that are redundant with respect to all other inequalities not containing variables other than x_i and x_j.

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

(v) 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)**–**(v)**, 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 $\alpha = x_n^*$, 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* $O(mn^2 \log m)$. $\square$

Assuming equal discounting factors does not substantially improve the running time of Algorithm 2. The Fourier-Motzkin elimination introduces discounting factors λ^k with $k > 1$, leaving us with a general MD2-system.

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 resulting 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 *are the number of vertices and edges of player* $\pi \in \{\min, \max\}$. $\square$

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 and optimal strategies of this DPG can be done as explained in the previous section. The MPG optimal strategies and values may be recovered from the DPG 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[7] asymptotically over the feasibility algorithm [7] for arbitrary 2-variable constraints. Designing new, more efficient algorithms to narrow the existing gaps, or establishing 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.

[7] Actually, it does not exploit discountedness.

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, 1988. Available online at: http://ioe.engin.umich.edu/people/fac/books/.

[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.