DIMACS Technical Report 2005-05
February 2005

Linear Complementarity Algorithms for Mean Payoff Games

by

Henrik Björklund, Ola Svensson, Sergei Vorobyov [1] ,[2] ,[3] ,[4] ,[5]

[1]DIMACS visitors, November, 2004.
[2]Uppsala University, Information Technology Department, Box 337, 751 05 Uppsala, Sweden.
[3]Supported by the Grant IG2003-2 067 from the Swedish Foundation for International Cooperation in Research and Higher Education (STINT, `www.stint.se`) and the Swedish Research Council grants "*Infinite Games: Algorithms and Complexity*" and "*Interior-Point Methods for Infinite Games*".
[4]DIMACS hosts: Leonid Khachiyan and Endre Boros
[5]Email: `vorobyov@csd.uu.se`

ABSTRACT

We suggest new pseudopolynomial and subexponential algorithms for *Mean Payoff Games* (MPGs). The algorithms are based on representing the MPG decision problem in the forms of non-standard and standard *Linear Complementarity Problems* (LCPs):

$$\text{find } w, z \geq 0 \text{ satisfying } w = Mz + q \text{ and } w^T \cdot z = 0, \tag{1}$$

and monotonic iterative propagation of slack updates.

Contents

1 Introduction

Mean Payoff Games (MPGs) [6, 7, 14, 13, 3] are full-information infinite adversary games on finite directed leafless[1] edge-weighted graphs $G = (V, E, w)$, with $V = V_{\max} \cup V_{\min}$ and $w : E \to \{-W, \ldots, W\} \subset \mathbb{Z}$. Starting in a vertex v_0 players Max and Min select edges (if $v_i \in V_{\max}$ then Max selects an edge from v_i and moves to its destination vertex v_{i+1}, otherwise Min chooses) and as a result construct an infinite sequence of edges (v_i, v_{i+1}). Player Max/Min wants to maximize/minimize, respectively,

$$\liminf_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} w(v_i, v_{i+1}), \quad \text{and } \limsup_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$

It turns out that MPGs are solvable in positional strategies for both players, every vertex has a value, which both players can secure by applying these strategies, and consequently the problem whether this value is above/below a certain threshold is in NP $\cap$ coNP [6, 7]. In this paper we concentrate on the *0-mean partition problem*: find a vertex subset $V^+ \subseteq V$ from which Max can secure a positive value. Other MPG problems, including finding exact values, ergodic partitioning, optimal strategies, are reducible to 0-mean partitioning, using dichotomy, edge reweighting, approximation [7, 3].

The previous known algorithms for solving MPGs (finding strategies and values) are as follows. 1) An exponential in the number of vertices algorithm implicit in [6]. 2) A pseudopolynomial (in the largest absolute edge weight) based on potential transformations [7, 13]. 3) A pseudopolynomial based on dynamic programming and another one based on fixpoint iteration of a contractive mapping [14]. 4) A strongly subexponential in the number of vertices and simultaneously pseudopolynomial (in the largest edge weight), based on the new *Longest-Shortest Paths* problem and randomized combinatorial linear programming [3].

In this paper we suggest new LCP-based algorithms. LCP stands for the *Linear Complementarity Problem* (1) [4, 8, 5], a well-investigated and powerful framework with numerous applications. The possibility of fitting MPGs in the LCP framework stems from the fact that it is possible to express the max and min functions using auxiliary slack variables subject to linear and orthogonality constraints. To our knowledge, the appealing opportunity of applying LCP-techniques to MPGs was never explored before. Besides the immediate application to MPGs, the novel contribution of our paper to the LCP-theory consists in showing that the particular class of MPG-generated LCPs is solvable in *pseudopolynomial* (in the values of the right-hand sides of linear constraints) and *randomized subexponential* (in the number of variables and constraints) time. It appears that the usual LCP complexity results are either polynomial or NP-hard. For the class of the so-called P-matrices (all principal minors positive, coNP-complete property) the existence of polynomial time LCP-algorithms is a long-standing open problem, while showing that either in a given LCP instance the matrix M is not P-, or finding a (unique) solution is in (functional) NP $\cap$ coNP [11] (but apparently no known subexponential algorithms [12]). Similarly, the decision problem for MPGs is also

[1] no sinks, i.e., every vertex has an outgoing edge.

in NP $\cap$ coNP and no polynomial algorithms in view. In contrast to P-matrix LCPs, for MPGs we have subexponential algorithms [3, 2, 1], and another one in this paper.

Outline. The paper is organized as follows. In Section 2 we show how to represent the 0-mean partition problem for an MPG in the form of a *non-standard* LCP, convenient for our algorithms. In Section 6 we represent it in standard form (1), in order to take advantage of existing LCP-algorithms. Correctness of the reduction is justified in Section 3. We describe the basic algorithm propagating slack correction updates in Section 4, justify its correctness, and analyze running time. In Section 5 we describe a subexponential LCP/MPG-algorithm based on a randomized scheme for combinatorial linear programming.

This paper is independent of (and in some sense orthogonal) to our [3, 2, 1], where we introduced and investigated the new "*controlled*" versions of the old optimization problems, like *shortest paths* and *linear programming*, nicely applicable for a wide range of games.

2 Representing Mean Payoff Games as Linear Complementarity Problems

For simplicity of explanations and to avoid notational clutter we make the following global

Assumption: Every MPG is

1. *binary*, i.e., the outdegree of every vertex is at most two,
2. *bipartite*, with partitions $V_{\max}$ and $V_{\min}$,
3. and *does not contain zero-weight loops*.

These assumptions are (almost) without loss of generality. Conditions #1 and #2 can be achieved by modeling k-ary vertices using binary and introducing intermediate vertices of the opponent. Although the means of the loops change, their signs are not, which is inconsequential for 0-mean partitioning. Achieving #3 can be done by multiplying edge weights by $|V| + 1$ and subtracting 1. Our algorithms may be straightforwardly recast, with certain complicating notational details and increased number of cases, to non-bipartite, non-binary games[2].

Given a binary bipartite MPG G without zero-weight loops, let us transform it into an LCP instance in three stages.

Stage One: Adding Sinks and Retreat Edges. We add two distinguished vertices to the game graph, called the *0-sink* and the *M-sink*, for a positive M selected as in (5). Besides, we add *retreat edges* from all vertices of Max to the 0-sink, and from all vertices of Min to the M-sink. Retreat edges have weight 0.

After this transformation every node will have outdegree ≤ 3, and the previously leafless game graph gets two sinks. The intuition behind adding sinks is as follows. If Max feels that he is unable to block negative weighted loops (accumulating $-\infty$), he would rather break any such loop by going to the 0-sink. Similarly, if Min cannot break a positive-weight loop, he prefers to lose a big M rather than $+\infty$ amount accumulated on any such loop, and prefers to retreat. Actually, after adding sinks neither Max can enforce positive, nor Min can enforce negative loops, and each play ends up in a sink.

Stage Two: Writing Linear Constraints for Slacks. We associate a *value* variable with every vertex of the MPG (sinks do not need variables), and a *slack* variable with every edge of the graph, including retreat edges. Throughout the paper we employ the following

[2] In which every loop contains (wlog) vertices of both players.

Notational Conventions. We use indexed variables x, y, u, v for vertex value variables, and s, l, r for slack variables. The slack variable associated to the edge from u to v (v not a sink) is indexed by $_{uv}$, while the weight of this edge is denoted w_{uv}. The slack variable for the edge from a vertex u to the sink is indexed $_u$. □

The intuition behind introducing slacks consist in expressing the *min* and *max* functions by linear constraints and *complementarity* conditions. We proceed as follows.

For every $v_i \in V_{\max}$ with successors $v_j, v_k \in V_{\min}$ we write the following linear equalities (maybe just two of them, if v_i has one successor v_j):

$$\begin{aligned} v_i &= s_i, \\ v_i &= v_j + w_{ij} + s_{ij}, \\ v_i &= v_k + w_{ik} + s_{ik}. \end{aligned} \tag{2}$$

Similarly, for every $v_i \in V_{\min}$ with successors $v_j, v_k \in V_{\max}$ we write

$$\begin{aligned} v_i + s_i &= M, \\ v_i + s_{ij} &= v_j + w_{ij}, \\ v_i + s_{ik} &= v_k + w_{ik}. \end{aligned} \tag{3}$$

Next, we add *nonnegativity constraints* $s_i \geq 0$ for all slacks, and denote the resulting linear constraint system by $LP(G)$.

Stage Three: Stipulating Complementarity. The system $LP(G)$ does not exactly express the semantics of *min* and *max* operators. For example, $v_i = s_i = 5$, $v_j = v_k = 3$, $s_{ij} = s_{ik} = w_{ij} = w_{ik} = 1$ satisfy (2), but $5 \neq \max(3{+}1, 3{+}1, 0)$. Similarly, (3) does not imply that $v_i = \min(v_j + w_{ij}, v_k + w_{ik}, M)$. To adequately express $v_i = \max(v_j + w_{ij}, v_k + w_{ik}, 0)$ by (2) and $v_i = \min(v_j + w_{ij}, v_k + w_{ik}, M)$ by (3) we finally add *slack complementarity constraints*

$$s_{ij} \cdot s_{ik} \cdot s_i = 0 \tag{4}$$

for each vertex v_i with successors v_j and v_k, which stipulate that in every triple of slacks at least one slack should be zero[3] (of course, for a vertex of outdegree one we write just $s_{ij} \cdot s_i = 0$). The resulting constraint system will be denoted *LCP*(G). Theorem 3.1 shows that *LCP*(G) is adequate for finding the 0-mean partition of G.

Remark. We will further refer to *LCP*(G) obtained from the MPG G as an instance of the *Linear Complementarity Problem* (*LCP*) [8]. Although our *LCP*(G) is not presented in *standard* form (1), we feel that the "linear complementarity" is essential in (2)–(4). Our algorithms do not require the standard form and, as an advantage, allow for the *generalized complementarity* for three or more variables; see (4). This is important (and more efficient) when dealing with games of arbitrary outdegree. We describe the reduction from MPGs to the standard form LCPs in Section 6. □

[3]Unfortunately, the function $\sum s_i \cdot s_{ij} \cdot s_{ik}$ is non-convex; otherwise we would immediately have a polynomial algorithm.

3 Correctness of Reduction from MPGs to LCPs

Now we prove the main theorem underlying our algorithms and describing the relation between the MPG and the solutions to the corresponding LCP. The proof is based on the *potential transformation* method and explains the choice of the Min's sink value M.

Theorem 3.1 *Every solution to LCP(G) solves the 0-mean partition problem for G.*

Proof. Let the value of Max's sink be 0, and the value of Min's sink equal

$$M = 1 + \sum\{\text{weights of the } n-1 \text{ heaviest (in absolute weights) edges}\}, \tag{5}$$

where we consider only edges of the original MPG.[4]

Assume we have a solution x^* to $LCP(G)$. From this solution we can easily find the 0-mean partition as follows. Start from a vertex v and follow zero slack edges (when there are several choices, select the leftmost one, assuming that the retreat edges are always rightmost). Notice that since there are no zero-weight loops, there can be no cycles on which all edges have slack 0. Thus we will eventually end up

1. either in the 0-sink, in which case v is in $V_{<0}$, the < 0 partition of G,

2. or in the M-sink, in which case v is in $V_{>0}$, the > 0 partition of G.

(Note that exactly one of the two should happen for each vertex v.)

To prove this, let us make a *potential transformation* of the edges of the game graph G' (G with added retreats) by transforming the weight w of each edge $u \to v$ as follows:

$$w'_{uv} = w_{uv} - u^* + v^*,$$

where u^*, v^* are values assigned to vertices u, v in the solution x^*.

Note that after this potential transformation:

1. every Max edge $u \to v$ gets weight $w'_{uv} = -s^*_{uv}$, i.e., the negated value of the slack assigned in x^* to this edge. Indeed, $u^* = w_{uv} + v^* + s^*_{uv}$, hence $w_{uv} - u^* + v^* = -s^*_{uv}$;

2. every Min edge $u \to v$ gets weight $w'_{uv} = s^*_{uv}$, the value of the slack assigned in x^* to this edge. Indeed, $u^* + s^*_{uv} = w_{uv} + v^*$, hence $w_{uv} - u^* + v^* = s^*_{uv}$.

Therefore, after the potential transformation, all Max's edges become non-positive, all Min's edges become non-negative, and zero-weight edges correspond exactly to zero slacks in x^*. Note that a potential transformation *preserves the total weight of any loop.* Thus 0-weight loops do not appear (since they were absent).

[4] It is better to choose $M = 1 + \max(M^+, -M^-)$, where M^+ is the maximal and M^- is the minimal sum of weights of at most $n-1$ edges of G. Alternatively, we could have symmetrically given Min's sink value 0, and Max's sink value $-M$.

Now throw away retreat edges from the game graph, start from any vertex in $V_{>0}$, let Max stick to his zero-weight edges, and allow Min to use arbitrary edges (all of them are non-negative). Since the graph is leafless (we deleted all sinks), every play is infinite and consists of the sum of positive loops (recall that zero loops are eliminated in the beginning). Therefore, the mean value of any such play is > 0, as needed, no matter which positional or non-positional strategy Min chooses.

Similarly, when the play starts in a vertex from $V_{<0}$, then Min can stick to his zero-slack edges (none of which goes to the sink). No matter what Max does he can only use non-positive edges and any loop that results is negative (again, there are no zero loops). Therefore, the mean value of any such play is < 0, as needed.

It remains to explain why we have to select the specific value of M described above. Let us show that from a Min vertex u taking a retreat (its value in x^* equals M) player Min cannot reach any Max vertex v that takes the retreat (its value in x^* equals 0). Suppose the contrary, and there exists a simple[5] path π from u to v on which Max only uses zero-slack edges. After the potential transformation the cost of the path $0 \leq c'(\pi) = c(\pi) - u^* + v^* = c(\pi) - M + 0$ (by telescoping). Hence, $c(\pi) \geq M$. But this is impossible, because π consists of at most $n-1$ edges and by the choice of M (5).

Similarly, we prove that from a Max vertex u taking a retreat (its value in x^* equals 0) player Max cannot reach any Min vertex v that takes the retreat (its value in x^* equals M). Suppose the contrary, and there exists a simple path π from u to v on which Min only uses zero-slack edges. After the potential transformation the cost of the path $0 \geq c'(\pi) = c(\pi) - u^* + v^* = c(\pi) - 0 + M$ (by telescoping). Hence, $c(\pi) \leq -M$. But this is impossible. These two contradictions prove that the partition of game vertices into $V_{>0}$ and $V_{<0}$ is correct and show that the choice of M is adequate. □

[5] we can throw away loops from a non-simple one

4 The Algorithm

Our algorithm starts by solving, in polynomial time, the system of linear constraints $LP(G)$ defined in Section 2, disregarding the complementarity constraints (4). The found feasible solution is then stepwise modified, by propagating "*slack updates*", producing new feasible solutions, which satisfy *more and more* complementarity conditions (4). Finally, when we have a feasible solution, which also satisfies complementarity, the problem is solved.

4.1 Iterative Slack Updates

At every iteration we select an *initial* vertex v that violates the complementarity condition, i.e., all outgoing slacks from v are *positive.* If there is no such vertex an LCP-solution and the MPG 0-mean partition has been found. When the initial vertex belongs to Max, all vertex values in the slack updates during this iteration may only *decrease* (monotonicity!), and we call the stage *decreasing.* Respectively, when the initial violating vertex is Min's, vertex values may only *increase* and the stage is called *increasing.* Note that in both stages slacks may decrease or increase.

We start by *decreasing* all three[6] slacks from the initial violating vertex v by the minimum of the three. This immediately corrects the slack complementarity in v[7], the value of v changes appropriately, and we propagate the update upward, through incoming edges of v, as follows. Suppose we changed the slacks and the value of vertex v. Looking at all edges entering v and at their slacks, we decide, whether to propagate or stop slack updates.

1. Suppose $v \in$Min, so its value increases, say by d.
2. Assume, an incoming edge (from a Max vertex u) has slack $s \geq d$.
3. Then we can set $s := s - d$ and terminate propagation along this edge, because this does not introduce any new violations, and may actually eliminate one if $s = d$.
4. If $d > s$ we need to propagate the value change upward, recursively. The exact procedure for these updates is described in Section 4.2.

When the stage terminates, we have at least one slack violation less, and we repeat the above while there are remaining violations.

Critical Properties. The algorithm has the following properties, which ensure termination, correctness, and the pseudopolynomial worst-case bound:

1. During propagation, *switches* may occur: some vertex switches its zero slack from one edge to another.

[6]or more, our algorithm easily deals with arbitrary graph degree.
[7]although propagating slack updates may return later back to v.

2. Every update stage terminates. Since we have constraints $v = s_v$ for $v \in V_{\max}$ and $u = M - s_u$ for $u \in V_{\min}$, and all slack variables have positive values, we know that no Max vertex can have a value below 0 and no Min vertex a value above M. The update stages either monotonically increase or monotonically decrease values, in integer steps. Since the graph is assumed to be bipartite, this ensures termination.

3. Crucial property: every update stage upon termination eliminates one violation and does not introduces any new violations. This can be seen from the propagation rules in Section 4.2, and ensures that the algorithm terminates in linearly many stages.

4. Since M depends linearly on the largest absolute edge weight and the number of vertices, the algorithm is *pseudopolynomial*.

4.2 Propagation rules

Let us specify the slack correcting propagation more formally. Recall that when the initial violation node belongs to Min, then all vertex values may only monotonically increase (increase-type stage), and when this node is Max, they may only decrease (decrease-type stage).

Propagation to a Min vertex. Suppose that the vertex value of a successor v_1 of a Min vertex v has been changed by c; see Figure 1.

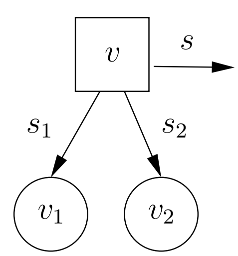

Figure 1: A Min node with the retreat and its slack variables

1. If $c > 0$ (increase-type stage), then

   (a) if $s_1 > 0$, then set $s_1 := s_1 + c$ and stop propagation along the edge marked s_1 (it was not tight in determining the value of the vertex v; by stopping we do not introduce any new violations);

   (b) if $s_1 = 0$, then let

   $$d = \min(c, s_2, s).$$

   Set $s_1 := c - d$, $s_2 := s_2 - d$, and $s := s - d$. Continue propagating d upwards from v. Notice that at least one slack from v will still be 0. If this is not s_1, we say that a *switch* has occurred.

2. If $c < 0$ (decrease-type stage) then

   (a) if $s_1 + c \geq 0$, then set $s_1 := s_1 + c$ and stop propagation along the edge marked s_1 (it was not tight in determining the value of the vertex v and have enough slack for not pushing its value up; by stopping we do not introduce any new violations, but can eliminate one, if s_1 becomes 0);

   (b) else if $s_1 + c < 0$, let $d = -s_1 - c$, increase the other two slacks by d, set $s_1 := 0$, and continue propagating d upwards from v.

Propagation to a Max vertex. Suppose that the (implicit) vertex value of a successor v_1 of a Max vertex v has been changed by c; see Figure 2.

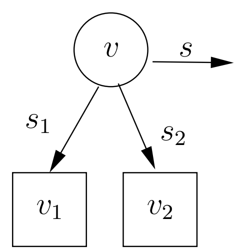

Figure 2: A Max node with the retreat and its slack variables

1. If $c > 0$ (increase-type stage) then

   (a) if $s_1 - c \geq 0$, then $s_1 := s_1 - c$, stop propagation along the edge marked s_1;

   (b) else if $s_1 - c < 0$, then let $d = c - s_1$, set $s_1 := 0$, increase the other two slacks by d and continue propagating d upwards from v.

2. If $c < 0$ (decrease-type stage) then

   (a) If $s_1 > 0$, then set $s_1 := s_1 - c$ and stop propagation along the edge marked s_1;

   (b) if $s_1 = 0$, let
   $$d = \min(-c, s_2, s),$$
   set $s_2 := s_2 - d$, $s := s - d$, and $s_1 := -c - d$. Continue propagating $-d$ upwards from v.

Theorem 4.1 *The slack propagation algorithms terminates in linearly many (in the number of vertices) stages, every stage is pseudopolynomial (in the maximal edge weight).* □

5 A Subexponential Algorithm

Let us now describe a subexponential, in the number of vertices of player Max, algorithm for solving an MPG-generated LCP. This is an advantage over a pseudopolynomial algorithm of Section 4 when edge weights are large. In this section we do not need to assume that the game is binary, i.e., the algorithm generates on an arbitrary outdegree graph without the preliminary transformation. This gives an advantage in complexity, as explained below.

It is convenient to identify edges with facets. A *facet* of the game is obtained when one fixes an edge from some vertex, throws away all other edges from that vertex, and allows all other edges vary, as in the original game. The algorithm proceeds as follows.

1. Generate an initial strategy σ for Max (randomly or arbitrarily). It consists in choosing exactly one edge leaving every vertex of Max, or one equation per Max variable in the LCP.

2. Pick uniformly at random any Max's edge e *not used* by σ, temporarily throw away this edge (and the facet G_e), and recursively find the "best"[8] strategy σ^* in the resulting subgame (subsystem) G'_e, i.e., a solution x^* satisfying complementarity conditions.[9] When the bottom of the recursion is hit and there remain no more choices for Max,[10] we are left with a graph/problem where only Min has choices. This one is easily solvable in polynomial time. Optimal solution for all vertices are provided by solving the shortest paths problem from all vertices to the sinks. Some values may be $-\infty$, some $+\infty$. The former give rise to immediate switches to the 0-sink, the latter are excluded from the further consideration, as definitely winning for Max, i.e., belong to V^+.

3. Look at the value v^* of v in x^* and at the value of the successors of v via the edge e, which was temporarily thrown away.

   (a) If this successor u has a value u^* (in x^*) such that $w_{vu} + u^* > v^*$, then *switch* in σ^* the edge originating from v to the edge e (such a switch is called *attractive*). Denote the resulting strategy σ and proceed to step 2.

   (b) otherwise, stop reporting σ^* as an optimal strategy.

Theorem 5.1 *This algorithm has expected running time $2^{O(\sqrt{n \log n})}$ and simultaneously deterministic $O(poly \cdot M)$, where $n = V_{\max}$ and M is defined by* (5).

We say the strategy is the best, because Max cannot increase the sum of all vertex variables (unless he switches to the edge e, whenever it is attractive). This follows easily from the potential transformation argument given in the proof of Theorem 3.1. On the other hand, if Max makes an attractive switch, it follows by the slack propagation rules defined in

[8]We will shortly explain why we say the "best" here.

[9]The strategy is easy to reconstruct once we know this solution: just follow edges assigned zero slacks.

[10]We decrease by one Max edge/choice at each level.

Section 4.2, that exactly one slack violation is introduced, which can be eliminated by upward slack updates. Moreover, these updates only increase the values of vertex variables, i.e., the switch is *profitable*, and the algorithm *monotonically increases the sum of all variables*. It follows that the algorithm never switches away from a facet it found attractive and switched to. Therefore, the subexponential analysis using hidden dimensions arguments due to [10] (for their version of the combinatorial linear programming algorithm) applies, and we obtain a randomized *subexponential*, in the number of vertices, LCP-algorithm for mean payoff games. It is simultaneously pseudopolynomial, as the previous one, due to a monotonic increase of the sum of all variables and natural bounds on this sum. Note that reducing the game to a binary one is a bad idea, since it may introduce quadratically many new vertices, and the algorithm becomes *exponential* (as Ludwig's). As another advantage of our algorithm, we never need to solve any linear programs.

Notice that we do not actually propagate slack corrections after the switch (because it may take pseudopolynomial time). We only use the slack updates to justify monotonicity of switches and use it to conclude that after finding the best strategy in a subgame and switching away to the complementary facet, the algorithm never leaves this facet. This is crucial for the subexponential analysis.

6 Representing MPGs by Standard LCPs

For the LCP problem in *standard* form (1) with square matrix, there is a multitude of known results, methods, and algorithms [8, 5], many of those depend on the structure of the matrix A. Our algorithms described above do not operate on standard form and do not exploit the special structure of the constraint matrix for slacks. It is interesting and important to analyze classes of matrices arising in reductions to the standard LCP from MPGs and other games. In particular, whenever such matrices happen to be PSD (positive semidefinite), there are polynomial LCP methods for solving them. Another possibility would be to represent MPGs as generalized LCP [4], which we will report elsewhere.

In order to express the 0-mean partition problem for MPGs in the *standard* LCP form, we need to make certain transformations and algebraic manipulations described in this section.

We still assume that the game is bipartite and binary. After adding sinks it become ternary. The transformations we make for Max and Min vertices are shown in Figure 3 and 4 [11]. The resulting game becomes binary but not bipartite: each player can either move to a sink or make two moves in a row. Binary degree is needed for expressing slacks complementarity (this will not be needed for generalized LCPs [4]). Note that each vertex is duplicated and two 0-weight edges are added. Thus the game graph increases only linearly.

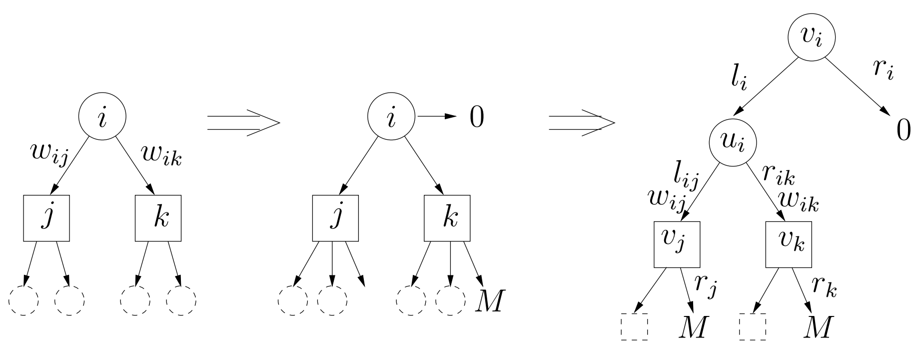

Figure 3: Transformation of a Max vertex

Our goal is to express left slacks via right slacks and to say that left and right slacks are complementary. Note our notational conventions: we denote left and right slacks by indexed l's and r's, and keep indexing consistent with vertex indexing in the original game graph.

Lemma 6.1 *For every vertex i (in the original game graph) with successors j, k via edges with weights w_{ij}, w_{ik}, introduce four slack variables, two left, l_i, l_{ij}, and two right, r_i, r_{ik}.*

If n is the number of vertices of the original game graph, then the $2n$ left slacks can be expressed through $2n$ right slacks via the standard LCP linear constraint system of the form $l = Ar + w$ as follows.

[11] We do not have to perform these transformations for vertices of outdegree 1, just adding retreats suffices. However, for the uniformity of explanations below, we first duplicate single edges from vertices of outdegree one.

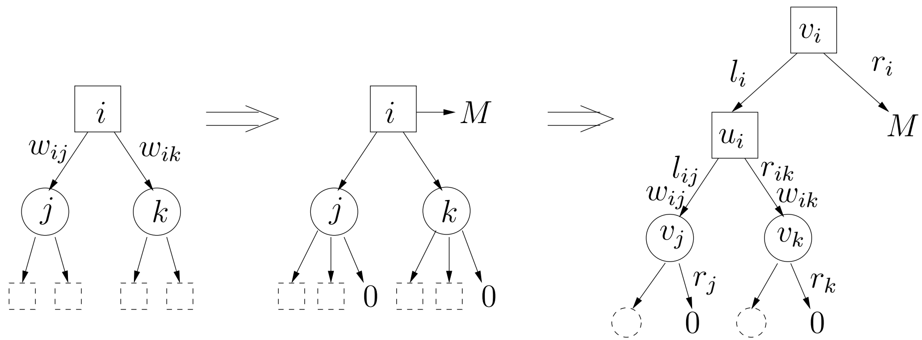

Figure 4: Transformation of a Min vertex

For every Max variable i with successors j, k write constraints:

$$l_i = r_i - r_{ik} + r_k - (w_{ik} + M), \quad (6)$$
$$l_{ij} = r_{ik} + r_j - r_k + (w_{ik} - w_{ij}). \quad (7)$$

For every Min variable i with successors j, k write constraints:

$$l_i = r_i - r_{ik} + r_k + (w_{ik} - M), \quad (8)$$
$$l_{ij} = r_{ik} + r_k - r_j + (w_{ik} - w_{ij}). \quad (9)$$

The complementary constraints are for every vertex i with successors j, k:

$$l_i \cdot r_i = 0, \qquad l_{ij} \cdot r_{ik} = 0, \quad (10)$$

We can ensure the following variables order in the vectors l, r of the LCP constraint system $l = Ar + w$: 1) the slack variables corresponding to Max variables precede the ones of Min, 2) variables l_i, l_{ij}, as well as r_i, r_{ik} occupy adjacent positions, and 3) l_i has the same position in l as r_i in r. Then complementarity constraints (10) can be written as a scalar product of two vectors $l^T \cdot r$, as needed, and matrix A has the following structure:

1. *$A[i, i] = 1$ and $A[i, i+1] = -1$ for odd-indexed lines $i = 2j - 1$ and $j = 1, \ldots, n$;*

2. *every line of A:*

   - either has exactly three nonzero coefficients per row, one off-diagonal +1 and one off-diagonal -1; this is the case for all odd-indexed lines $i = 2j - 1$ with $j = 1, \ldots, n$;

   - or has a unique +1 on the main diagonal; this happens in even-indexed lines $2i$, if vertex i has two coinciding successors $j = k$; [12]

3. *the remaining nonzero entries $A[i, j] \neq 0$ are located in gray areas, as shown in Figure 5.*

[12]Recall that for uniformity we duplicate a single edge from a unary vertex, so as it becomes binary with the same left and right successors.

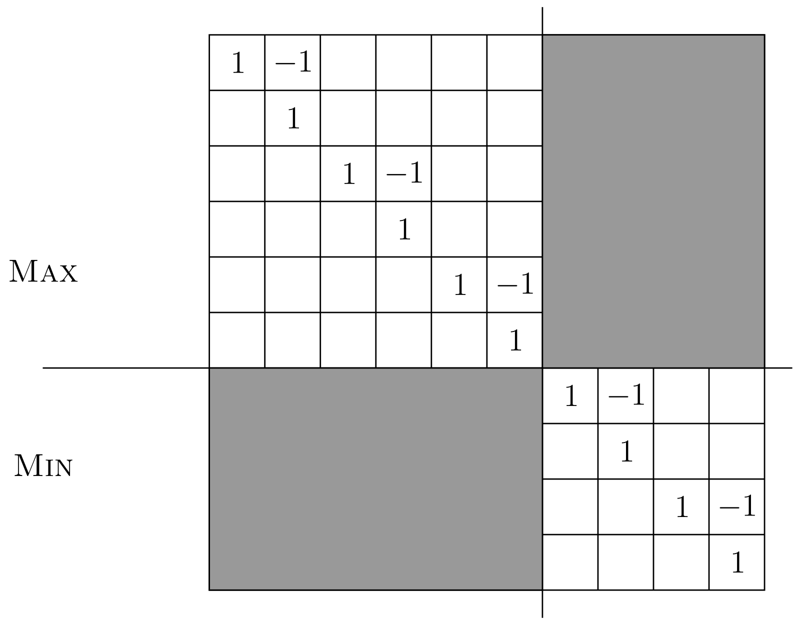

Figure 5: Structure of the matrix A in the standard LCP $l = Ar + w$ arising from an MPG

Proof. For every Max vertex v_i and its duplicate u_i (see Figure 3) we write the system of constraints

$$\begin{aligned} v_i &= u_i + l_i, && (11) \\ v_i &= r_i, && (12) \\ u_i &= w_{ij} + v_j + l_{ij}, && (13) \\ u_i &= w_{ik} + v_k + r_{ik}. && (14) \end{aligned}$$

Note that the values of the Min vertices v_j, v_k (see Figure 3) equal $v_j = M - r_j$, $v_k = M - r_k$. Therefore, we obtain (6):

$$\begin{aligned} l_i &= r_i - u_i && \text{(by (11) and (12))} \\ &= r_i - (w_{ik} + (M - r_k) + r_{ik}) && \text{(by (14), using } v_k = M - r_k\text{)} \\ &= r_i - r_{ik} + r_k - (w_{ik} + M). \end{aligned}$$

We can express l_{ij} by equating (13) and (14), using also $v_j = M - r_j$, $v_k = M - r_k$, obtaining (7): $l_{ij} = r_{ik} + (w_{ik} - w_{ij}) + (v_k - v_j) = r_{ik} + r_j - r_k + (w_{ik} - w_{ij})$.

Symmetrically, for every Min vertex v_i and its duplicate u_i (see Figure 4) we write the system of constraints

$$\begin{aligned} v_i + l_i &= u_i, && (15) \\ v_i + r_i &= M, && (16) \\ u_i + l_{ij} &= w_{ij} + v_j, && (17) \\ u_i + r_{ik} &= w_{ik} + v_k. && (18) \end{aligned}$$

Note that the values of the Max vertices v_j, v_k (see Figure 4) equal $v_j = r_j$, $v_k = r_k$. Therefore, we get (8):

$$\begin{aligned} l_i &= r_i + u_i - M && \text{(by (15) and (16))} \\ &= r_i + (-r_{ik} + w_{ik} + r_k) - M && \text{(by (18), using } v_k = r_k\text{)} \\ &= r_i - r_{ik} + r_k + (w_{ik} - M). \end{aligned}$$

We can express l_{ij} from (17), (18), using also $v_j = r_j$, $v_k = r_k$, which gives (9):
$l_{ij} = r_{ik} + (w_{ik} - w_{ij}) + (v_k - v_j) = r_{ik} + r_k - r_j + (w_{ik} - w_{ij})$.

The remaining claims of the lemma follow by inspection. □

7 MPGs Generate LCPs with Non-P-Matrices

Finding a polynomial time algorithm for LCPs (1) with P-matrices (all principal minors positive) is a Holy Grail of the LCP-community. Interior point methods [9] fail, randomized pivoting fairs poorly [12]. Unlike our acyclic orientations of combinatorial cubes induced by MPG-generated LCPs, underlying our successful subexponential algorithm from Section 5, combinatorial cubes induced by LCPs with P-matrices may have loops, and no subexponential algorithms are currently known. One may wonder whether the LCP with P-matrices is more general than MPG-generated LCPs investigated in this paper. If the answer were positive, and P-matrix LCP were ever proved polynomial, this could boost the performance of the MPG-decision algorithms. Some LCP proponents erroneously suggested that our LCPs are a subclass of P-matrix LCPs. However,

Proposition 7.1 *Every MPG generates a non-P-matrix LCP.*

Proof. Consider a simple MPG depicted in Figure 6.

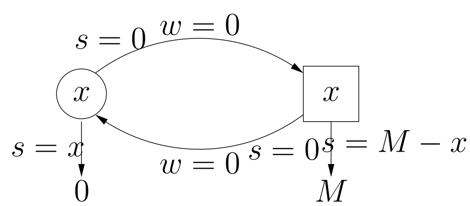

Figure 6: MPG inducing a non-P-matrix LCP

When writing the LCP for this game, as described in Section 2, all edge weights and sink values end up in the right-hand side vector q of the LCP (1). The well-known criterion is: M is a P-matrix iff the LCP has a *unique* solution for every vector q. Looking at the game in Figure 6, we notice that when edge weights are 0, and $M > 0$, the corresponding LCP has *many* solutions: vertex and slack values as indicated, for any $0 \leq x \leq M$.

Notice that this actually happens to *any* MPG, since it always has loops and we can produce a q yielding non-unique solutions as above by assigning all edges weight 0. $\square$

8 MPGs Generate Non-PSD Matrices

Recall that a square matrix A, whether symmetric or not, is called *positive semidefinite* (PSD) if $x^T Ax \geq 0$ for every x.

The classes of P- and PSD-matrices intersect (e.g., unit matrix is both), but there are P-matrices that are not PSD (e.g., $\begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}$, take $x = (1, -1)^T$), and PSD-matrices that are not P-, e.g., $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. Therefore it is interesting to figure out whether MPGs generate PSD-matrices.

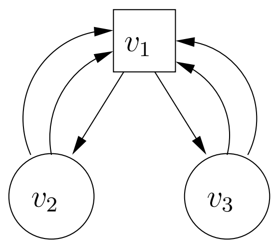

Figure 7: An MPG generating a non-PSD matrix LCP

Theorem 8.1 *MPGs generate, in general, non-PSD-matrix LCPs.*

Proof. According to Lemma 6.1, the MPG shown in Figure 7 generates the following LCP:

$$\begin{aligned}
l_1 &= r_1 - r_{13} + r_3 + c_1 \\
l_{12} &= r_{13} + r_2 - r_3 + c_2 \\
l_2 &= r_2 - r_{21} + r_1 + c_3 \\
l_{21} &= r_{21} + c_4 \\
l_3 &= r_3 - r_{31} + r_1 + c_5 \\
l_{31} &= r_{31} + c_6,
\end{aligned}$$

where c_i are constants depending on the weights of edges and sinks. The matrix of this system is

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

It is well known that if a matrix A is PSD, then all eigenvalues of the symmetric matrix $1/2(A + A^T)$ are nonnegative. But the minimal eigenvalue for matrix $1/2(A + A^T)$ is approximately $-0.26 < 0$. Thus A is not PSD. $\square$

9 Complexity Issues: NP $\cap$ coNP Membership and NP-Hardness

A useful consequence for our results and algorithms from the previous sections is the following

Corollary 9.1 *The problem of determining whether a vertex belongs to the ≤ 0 or > 0-mean partition of an MPG is in NP $\cap$ coNP, and can be solved in time subexponential in the number of game vertices and pseudopolynomial in the larges edge weight.* $\square$

Similar results hold, by reduction to the 0-mean partition [3], for all other MPG-problems, like decision versions of the vertex value problem, ergodic partitioning, etc.

MPG-generated LCPs *always* have solutions, and Corollary 9.1 shows that finding them is presumably not NP-hard.

NP $\cap$ coNP-membership (without subexponential algorithms though [12]) is also known for the P-matrix LCP [11], and NP-hardness is known for P_0-matrix LCPs.

We conclude by showing that an LCP with three ± 1 per row (the structure close to our Lemma 6.1) is NP-hard.

Theorem 9.2 *The LCP problem is NP-complete even for matrices with only three ± 1 elements per row.*

Proof. Let ψ be a 3CNF-formula with n variables. For each variable x in ψ add the constraint $l_x + r_x = 1$ and the complementarity condition $l_x r_x = 0$. For each clause c_i add complementarity $l_{n+i} r_{n+i} = 0$.

For a clause $c_i = L_{i1} \vee L_{i2} \vee L_{i3}$ define a linear polynomial $p(c_i) = m(L_{i1}) + m(L_{i2}) + m(L_{i3})$, where $m(x) = r_x$ and $m(\neg x) = 1 - r_x$. For each clause c_i in ψ add constraint $l_{i+n} + p(c_i) = 2$. Equivalently, for each clause c_i in ψ add one of the four constraints, depending on the number of variables negated:

$$\begin{aligned}
x \vee y \vee v \quad &\Rightarrow \quad l_{i+n} + r_x + r_y + r_v = 2, \\
\neg x \vee y \vee v \quad &\Rightarrow \quad l_{i+n} - r_x + r_y + r_v = 1, \\
\neg x \vee \neg y \vee v \quad &\Rightarrow \quad l_{i+n} - r_x - r_y + r_v = 0, \\
\neg x \vee \neg y \vee \neg v \quad &\Rightarrow \quad l_{i+n} - r_x - r_y - r_v = -1,
\end{aligned}$$

where r_x, r_y, r_v represent the variables x, y, v in the LCP.

Let us show that if ψ is satisfiable then the LCP has a solution. Let $\bar{v} = (v_1, v_2, \ldots, v_n)$ be a satisfying truth assignment for ψ. For each $v_i \in \bar{v}$, if $v_i = T$ then let the corresponding variables in the LCP have the values $l_i = 1$, $r_i = 0$, and $l_i = 0$, $r_i = 1$ if $v_i = F$. Notice that the variable assignments satisfies the complementarity condition in the LCP and for each c_i the value of $p(c_i)$ equals either 0, or 1, or 2, since every clause c_i is satisfied. Therefore, l_{n_i} can be assigned values ≥ 0.

Conversely, we have to prove that if the LCP has a solution then ψ is satisfied. Let (l, r) be a solution to the LCP. For each variable v_i we have that either $l_i = 1$ and $r_i = 0$, or $l_i = 0$

and $r_i = 1$. In the former case we set $v_i = T$, and $v_i = F$ otherwise. With this assignment the value of each $p(c_i)$ equals 0, 1, or 2, and this means that all c_i's, hence ψ are satisfied. $\square$

Reducing an MPG to a Standard Linear Complementarity Problem (SLCP), introduces a lot of useful methods and algorithms. However, the matrix in our reduction is not PSD, in general (Theorem 8.1), and it is easy to see that it is not a triangular or a Z-matrix (all off-diagonal entries are nonpositive). Thus there is no known polynomial algorithm for SLCP applicable to our problem, and Lemke's algorithm is exponential in the worst case [5]. Even though the matrix in our reduction from MPGs is sparse (at most three elements per row), by Theorem 9.2, this fact is hard to exploit. All these facts motivate our alternative approach described in Sections 2–5.

10 Conclusions

1. Our reduction from MPGs to the standard form LCPs produces very restricted matrices; see Lemma 6.1. Do any special matrices appear in reductions from MPGs (at least from some rich and interesting classes of MPGs) that allow for polynomial time LCP-algorithms [8, 5, 9]? Can these cases be efficiently distinguished by analyzing game graphs? Although we showed that MPG-generated LCPs are non-P-matrix, exploiting and adjusting appropriate LCP methods seems promising.

2. The performance of our basic algorithm, in the worst case, may match its pseudopolynomial upper bound (this is also true for the algorithms in [7, 14]). The behavior of the algorithm depends on the choice of the initial feasible solution for linear constraints, with complementarity disregarded. Further investigations are necessary to choose this solution cleverly, maybe by replacing a non-convex precise target function $\sum s_i' \cdot s_i'' \cdot s_i'''$ (actually, we know its min value 0) by some meaningful convex approximation. It would also be interesting to recast our graph-theoretic slack-update algorithm to a more standard algebraic pivoting setting and extend it to discounted payoff games and to simple stochastic games.

References

[1] H. Björklund, O. Nilsson, O. Svensson, and S. Vorobyov. Controlled linear programming: Boundedness and duality. Technical Report DIMACS-2004-56, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, December 2004. http://dimacs.rutgers.edu/TechnicalReports/.

[2] H. Björklund, O. Nilsson, O. Svensson, and S. Vorobyov. The controlled linear programming problem. Technical Report DIMACS-2004-41, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, September 2004. http://dimacs.rutgers.edu/TechnicalReports/.

[3] H. Björklund, S. Sandberg, and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. In J. Fiala, V. Koubek, and J. Kratochvil, editors, *29th International Symposium on Mathematical Foundations of Computer Science (MFCS'2004)*, volume 3153 of *Lecture Notes in Computer Science*, pages 673–685. Springer, August 2004. Full version to appear in *Discrete Applied Mathematics*. Preliminary full version: TR DIMACS-2004-05 http://dimacs.rutgers.edu/TechnicalReports/.

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

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

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

[7] 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.

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

[9] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise. *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems*, volume 538 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[10] J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[11] N. Megiddo. A note on the complexity of P-matrix LCP and computing the equilibrium. Technical Report RJ 6439 (62557) 9/19/88, IBM Almaden Research Center, 1988.

[12] W. D. Morris. Randomized pivot algorithms for P-matrix linear complementarity problems. *Mathematical Programming, Ser. A*, 92:285–296, 2002.

[13] N. Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.

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