A Polynomial Time Algorithm to Solve a Class of Optimization Problems with a Multi-linear Objective Function and Affine Constraints

Hadi Charkhgard\textsuperscript{a}, Martin Savelsbergh\textsuperscript{b}, and Masoud Talebian\textsuperscript{a}

\textsuperscript{a}School of Mathematical and Physical Sciences, The University of Newcastle, Australia
\textsuperscript{b}School of Industrial and Systems Engineering, Georgia Institute of Technology, USA

December 29, 2014

Abstract

We present the first polynomial-time linear programming based algorithm for a class of optimization problems with a multi-linear objective function and affine constraints. This class of optimization problems arises naturally in a number of settings in game theory, such as the bargaining problem, linear Fisher markets, and Kelly capacity allocation markets, but has applications in other fields of study as well. The algorithm computes an optimal solution by solving at most $O(p^3)$ linear programs, where $p$ is the number of variables in the multi-linear objective function.

Keywords: game theory, convex programs, multi-linear objective function, polynomial-time algorithm, linear programming

1 Introduction

In this paper, we study optimization problems of the form,

$$\max \prod_{i=1}^{p} y_i$$

s.t. $y = Dx + d$

$Ax \leq b$

$x, y \geq 0, \ x \in \mathbb{R}^n, \ y \in \mathbb{R}^p,$

with $D$ a $p \times n$ matrix, $d$ a $p$-vector, $A$ an $m \times n$ matrix, and $b$ an $m$-vector. Such optimization problems often arise in game theory settings where the $x$ variables represent players’ actions and the $y$ variables represent players’ utilities. Examples include computing the Nash solution to a bargaining problem [8, 9] and computing an equilibrium of a linear Fisher or a Kelly capacity allocation market [3, 2, 5, 11].

We will refer to such an optimization problem as a \textit{positive multi-linear program with affine constraints} (PMP-A). We refer to the set $\mathcal{X} := \{x \in \mathbb{R}^n : Ax \leq b, \ x \geq 0\}$ as the \textit{decision space} and to the set $\mathcal{Y} := \{y \in \mathbb{R}^p : x \in \mathcal{X}, \ y = Dx + d, \ y \geq 0\}$ as the \textit{payoff space}. We assume that $\mathcal{X}$ is

\textsuperscript{*}Corresponding author. Tel.: +61 424 607 237
E-mail address: hadi.charkhgard@uon.edu.au
bounded (which implies that \( \mathcal{Y} \) is compact) and that the optimal objective value of the problem is strictly positive, i.e., there exists a \( y \in \mathcal{Y} \) such that \( y > 0 \). We usually refer to \( x \in \mathcal{X} \) as a feasible solution and to \( y \in \mathcal{Y} \) as a feasible point. In fact, \( y \) is the map of \( x \) in the payoff space. We denote the optimal solution and the optimal point of a PMP-A by \( x^* \) and \( y^* \), respectively. Finally, we assume that \( x^* \) and \( y^* \) are rational.

A PMP-A can both be seen as an extension of a linear program and as a special case of a geometric programming problem. It can be shown that a PMP-A has a unique optimal point \([8]\). Moreover, it is easy to see \( x^* \) is an efficient (or Pareto optimal) solution, i.e., a solution in which it is impossible to improve the value of a player’s payoff without a deterioration in the value of at least one other player’s payoff, and, consequently, \( y^* \) is a nondominated point (the map of an efficient solution into the payoff space is called a nondominated point).

A convex program solver, e.g., one that uses an interior point method, can find an optimal solution to a PMP-A in polynomial time \([4]\). (Note that because we have assumed that the optimal value of the objective function is not zero, it is possible to use \( \sum_{i=1}^{n} \log y_i \) as the objective function). However, convex program solvers are significantly slower than linear programming solvers (such as the IBM ILOG CPLEX Optimizer, the FICO Xpress Optimizer, and the Gurobi Optimizer. Therefore, as observed in \([12]\), algorithms that exploit the power of linear programming solvers may have a significant computational advantage.

That is exactly what we do. The main contribution of our research is the development of a linear programming based (LP-based) polynomial-time algorithm for solving a PMP-A. A computational study shows that our algorithm significantly outperforms the well-known convex programming solver IPOPT on this class of problems.

As far as we know, our algorithm is the first polynomial-time LP-based algorithm for solving a PMP-A with an arbitrary number of variables in the multi-linear objective function. Vazirani \([12]\) presents an LP-based algorithm for solving a PMP-A with a bilinear objective function (i.e., with two variables in the objective function), the Binary Search Algorithm (BSA). Both our algorithm and BSA work in the payoff space, but in different ways. BSA searches the payoff space for a facet containing the optimal point (note that in two dimensional space, facets are line segments that are characterized by their endpoints) and, once found, computes the optimal point. Our algorithm searches the payoff space by iteratively removing sections that are guaranteed not to contain the optimal point. In the process, a lower and upper bound on the optimal objective value are continuously updated until they have converged to the same value.

We note that PMP-As do not only arise in game theory. At the end of the paper, we briefly mention some applications of PMPs-A in other fields of study, including geometry, statistical estimation, approximation, and polynomial programming.

The remainder of the paper is organized as follows. In Section 2, we introduce some well-known applications of PMPs-A in game theory. In Section 3, we detail the logic of our proposed algorithm. In Section 4, we report on the results of a computational study. In Section 5, we introduce applications of PMPs-A in other fields of study. Finally, in Section 6, we give some concluding remarks.

## 2 Positive multi-linear programs with affine constraints arising in game theory

We present three well-known game theoretic settings that give rise to PMP-As. We refer the interested readers to \([2, 3, 5, 11]\) for further information and more details.

### 2.1 Bargaining Problems

A bargaining problem is a cooperative game in which all players agree to create a grand coalition, instead of competing with each other, to get a higher payoff \([10]\). To be able to create a grand coalition, the agreement of all players is necessary. Therefore, a critical question to be answered
is: What should the payoff of each player be in a grand coalition? One of the solutions to the (symmetric) bargaining problem was proposed by Nash and is now known as the Nash bargaining solution [8, 9].

We start by explaining the Nash bargaining solution in the case if two players. We denote the expected utility values of the players by $y = (y_1, y_2)$. Let $\mathcal{Y}$ be the 2-dimensional feasible set in the payoff space containing all possible expected utility values of the players. We assume that $\mathcal{Y}$ is compact and convex, and that it is given to us by an oracle in the form of a set of affine constraints. Let $\mathcal{Y}_N$ be the nondominated frontier (i.e., the set of nondominated points) of $\mathcal{Y}$. Let the disagreement point $d := (d_1, d_2)$ in the payoff space represent the payoffs that the players will receive if they do not create the coalition.

Two classical axioms imposing restrictions on a solution to a pure bargaining problem are

- **Individual Rationality**: None of the players accepts a payoff lower than the one which is guaranteed to him under disagreement, i.e., $y_1 \geq d_1$ and $y_2 \geq d_2$.
- **Pareto Optimality**: The solution must be such that the payoff for one player cannot be increased without decreasing the payoff of the other player.

Let $\mathcal{Y}^*: = \{ y \in \mathcal{Y} : y_1 \geq d_1, y_2 \geq d_2 \}$ and $\mathcal{Y}_N^*: = \{ y \in \mathcal{Y}_N : y_1 \geq d_1, y_2 \geq d_2 \}$. To satisfy the classical axioms, a bargaining solution $y^*$ must be in $\mathcal{Y}^*_N$. However, in general, $\mathcal{Y}^*_N$ still contains an infinite number of points. Nash introduced three additional axioms:

- **Symmetry**: If $\mathcal{Y}^*$ is symmetric, i.e., for any vector $(y, y') \in \mathcal{Y}^*$, the vector $(y', y)$ is also in $\mathcal{Y}^*$, then in a bargaining solution we must have $y^*_1 = y^*_2$.
- **Linear Invariance**: Let $y^*$ be a solution to a bargaining game $G$. Moreover, let $\hat{G}$ be a bargaining game obtained from $G$ by an order-preserving linear transformation $T$ of one player’s utility function. The solution $\hat{u}^*$ to the bargaining game $\hat{G}$ has to be the image of $y^*$ under $T$, i.e., $\hat{u}^* = Ty^*$.
- **Independence of Irrelevant Alternatives**: Let $d$ be the disagreement point and $y^*$ be a solution to the bargaining game $G$. Moreover, let $\hat{G}$ be a bargaining game that is obtained from $G$ by restricting $\mathcal{Y}$ to $\hat{\mathcal{Y}}$, i.e., $\hat{\mathcal{Y}} \subset \mathcal{Y}$. If $d \in \hat{\mathcal{Y}}$ and $y^* \in \hat{\mathcal{Y}}$, then $y^*$ is the solution of $\hat{G}$.

Nash proved that under the above five axioms, the optimal solution $y^* = (y^*_1, y^*_2)$ to the bargaining problem is the unique point satisfying

$$y^* \in \arg \max \{ (y_1 - d_1)(y_2 - d_2) : y \in \mathcal{Y}, y_1 \geq d_1, y_2 \geq d_2 \}.$$

Nash’s result can be extended to pure bargaining problems with $p$ players ($p > 2$) straightforwardly. Let $\mathcal{Y}$ be the $p$-dimensional feasible set in the payoff space, then for a disagreement point $d = (d_1, \ldots, d_p)$, the optimal solution $y^* = (y^*_1, \ldots, y^*_p)$ to the bargaining problem is a point satisfying

$$y^* \in \arg \max \{ \prod_{i=1}^{p} (y_i - d_i) : y \in \mathcal{Y}, y_i \geq d_i \ \forall i \in \{1, \ldots, p\} \}.$$

Kalai [6] established that the Nash bargaining solution can be extended even further to non-symmetric bargaining games. He showed that by removing the Symmetry axiom, the resulting problem, i.e., the nonsymmetric bargaining game, must have a unique solution and this solution can be found by solving the following optimization problem,

$$y^* \in \arg \max \{ \prod_{i=1}^{p} (y_i - d_i)^{m_i} : y \in \mathcal{Y}, y_i \geq d_i \ \forall i \in \{1, \ldots, p\} \}.$$ 

where $m_i$ is a positive integer for all $i \in \{1, \ldots, p\}$. It is easy to see that Kalai’s optimization problem can be transformed to the (standard) Nash optimization problem by making $m_i$ copies of $y_i$ for all $i \in \{1, \ldots, p\}$. We denote these copies with $\hat{y}_{i,j}$ for all $i \in \{1, \ldots, p\}$ and for
all \( j \in \{1, \cdots, m_i\} \). As a consequence, the solution of a nonsymmetric bargaining game can be obtained by solving the following optimization problem,

\[
y^* \in \arg\max\{ \prod_{i=1}^{p} \prod_{j=1}^{m_i} (\hat{y}_{(i,j)} - d_i) : y \in Y, y_i = \hat{y}_{(i,j)} \forall i \in \{1, \cdots, p\} \forall j \in \{1, \cdots, m_i\}, y_i \geq d_i \forall i \in \{1, \cdots, p\}\}.
\]

Therefore, an algorithm that computes the Nash bargaining solution for a game with an arbitrary number of players can be used to compute the solution of nonsymmetric bargaining games too. Note that due to the fact that the Nash bargaining solution must be in the set \( Y_N^* \), we can always assume that the origin of Cartesian coordinate system is transferred to \( d \) (note that this can be done by using a simple linear transformation). As a consequence and without loss of generality, we can assume that \( d = (0, \cdots, 0) \). In summary, to compute the solution to a symmetric or nonsymmetric bargaining game, we need to solve a PMP-A. (In Section 3, we show that the transformation that is used to change a nonsymmetric game to a symmetric game can be exploited effectively by our proposed algorithm.)

### 2.2 Linear Fisher markets

In a linear Fisher market, we are given a market with \( n \) divisible items (without loss of generality, we assume that 1 unit is available of each item) and \( p \) agents. Each agent \( i \) has an initial endowment \( m_i \), where \( m_i \) is a positive integer for all \( i \in \{1, \cdots, p\} \), and has a linear utility function \( z_i(x) := \sum_{j=1}^{n} u_{ij} x_{ij} \), where \( x_{ij} \) is the fraction of item \( j \) given to agent \( i \) and \( u_{ij} \) is a nonnegative parameter.

The goal in a linear Fisher market is to compute a market equilibrium. More precisely, the goal is to compute the price \( w_j > 0 \) of each item \( j \) and a feasible allocation of items, i.e.,

\[
\sum_{i=1}^{p} x_{ij} \leq 1 \quad j \in \{1, \cdots, n\},
\]

such that the following conditions hold

- Every agent spends his initial endowment, i.e., \( \sum_{j=1}^{n} w_j x_{ij} = m_i \);
- All items with positive prices must be fully sold, i.e., if \( w_j > 0 \) then \( \sum_{i=1}^{p} x_{ij} = 1 \); and
- If an item \( j \) is allocated to an agent \( i \), then it must have the highest utility-to-price ratio. i.e., if \( x_{ij} > 0 \) then \( \frac{u_{ij}}{w_j} \geq \frac{u_{ij'}}{w_{j'}} \) for \( j' \in \{1, \cdots, n\} \).

Eisenberg and Gale [3] showed that to find a market equilibrium, the following convex optimization problem has to be solved,

\[
\max \prod_{i=1}^{p} y_{ni}^{m_i} \\
\text{s.t: } y_i = z_i(x) \quad \forall i \in \{1, \cdots, p\}, \\
\sum_{i=1}^{p} x_{ij} \leq 1 \quad \forall j \in \{1, \cdots, n\} \\
x_{ij} \geq 0 \quad \forall i \in \{1, \cdots, p\}, \forall j \in \{1, \cdots, n\}
\]

Observe that to compute a market equilibrium in a linear Fisher market a PMP-A has to be solved. (Using a transformation similar to the one used to transform a nonsymmetric bargaining problem to a symmetric bargaining problem.)

### 2.3 Kelly capacity allocation markets

In a Kelly capacity allocation market [7], an example of the broader class of resource allocation markets, there is direct network \( D(N, A) \) and there are \( p \) agents. Each arc \( a \in A \) of the network
has a capacity \(c_a\), each agent \(i \in \{1, \ldots, p\}\) has an initial endowment \(m_i \in \mathbb{Z}_{>0}\), and wants to send flow from a source node \(s_t \in N\) to a sink node \(t_i \in N\).

Similar to linear Fisher markets, the goal in a Kelly capacity allocation market is to compute a market equilibrium. More precisely, the goal is to compute the price \(w_a \geq 0\) of each arc \(a \in A\) and a flow \(y_i \geq 0\) for each agent \(i \in \{1, \ldots, p\}\) such that the following conditions hold

- Every agent spends his initial endowment;
- All arcs with positive prices must be fully saturated; and
- The flow of each agent is sent on a minimum priced path from source to sink.

Let \(X_i\) be the set of all paths from \(s_t\) to \(t_i\). Moreover, let \(X_i^a\) be the set of all paths from \(s_t\) to \(t_i\) that contains arc \(a\). Let \(f_i(x)\) where \(x \in X_i\) denote the flow on path \(x\) from \(s_t\) to \(t_i\). It can be shown that to find a market equilibrium, the following convex optimization problem has to be solved

\[
\begin{aligned}
\max & \quad \prod_{i=1}^{p} y_i^{m_i} \\
\text{s.t.} & \quad y_i = \sum_{x \in X_i} f_i(x) \quad \forall i \in \{1, \ldots, p\} \\
& \quad \sum_{i=1}^{p} \sum_{x \in X_i^a} f_i(x) \leq c_a \quad \forall a \in A \\
& \quad f_i(x) \geq 0 \quad \forall i \in \{1, \ldots, p\}, x \in X_i \\
\end{aligned}
\]

Observe that to compute a market equilibrium in a Kelly capacity allocation market a PMP-A has to be solved.

### 3 A Polynomial Time Algorithm

We first introduce concepts and notation that will facilitate the presentation and discussion of our algorithm. We use the notation \(\mathbb{R}^p_{\geq} := \{y \in \mathbb{R}^p : y \geq 0\}\) for the nonnegative orthant of \(\mathbb{R}^p\), and \(\mathbb{R}^p_{>} := \{y \in \mathbb{R}^p : y > 0\}\) for the positive orthant of \(\mathbb{R}^p\). Let \(z^1, z^2 \in \mathbb{R}^p_{>}\) be two points in the payoff space. We denote by \(d(z^1, z^2)\) the Euclidean distance between these points in the payoff space. Let \(z \in \mathbb{R}^p_{>}\) be a point in the payoff space. We denote by \(B(z)\) the box in the payoff space defined by the origin and \(z\), and by \(v(B(z))\) its (hyper)volume. We denote by

\[
L(z) := \{(y_1, \ldots, y_p) \in \mathbb{R}^p : \frac{y_1}{z_1} = \frac{y_2}{z_2} = \cdots = \frac{y_p}{z_p}\},
\]

the line defined by the origin and \(z\), and we denote by

\[
H(z) := \{(y_1, \ldots, y_p) \in \mathbb{R}^p : \sum_{i=1}^{p} \frac{y_i}{z_i} = 1\},
\]

the hyperplane defined by the points \((z_1, 0, \ldots, 0), (0, z_2, 0, \ldots, 0), \ldots, (0, \ldots, 0, z_p)\). Our algorithm uses cutting planes to restrict the set \(\mathcal{Y}\). For two point \(z, z' \in \mathbb{R}^p_{>}\), we denote by

\[
C(z, z') := \{(y_1, \ldots, y_p) \in \mathbb{R}^p : \sum_{i=1}^{p} \frac{y_i}{z_i} \geq \sum_{i=1}^{p} \frac{y_i'}{z_i'}\},
\]

the cutting plane defined by \(z\) and \(z'\). We denote by \(L_c\) the list of generated cutting planes, and by \(\mathcal{Y}(L_c)\) the set of feasible solutions in the payoff space after adding the cutting planes in \(L_c\).

An intersection point \(z^l \in \mathbb{R}^p\) of \(L(z)\) with the common boundaries of \(\mathcal{Y}(L_c)\) and \(\mathcal{Y}\) can be obtained by solving the optimization problem

\[
z^l = \arg \max \{\sum_{i=1}^{p} y_i : y \in \mathcal{Y}(L_c), y \in L(z)\}.
\]
We denote this search by \textsc{Find-Intersection}(z). Observe that to obtain \( z^I \) only a single LP has to be solved.

We often use an upper bound point \( z^u \in \mathbb{R}^p \) with \( z^u_i = \max_{y \in \mathcal{Y}(L_c)} y_i \) for all \( i \in \{1, \ldots, p\} \). We denote the operation of computing this point for a set \( \mathcal{Y}(L_c) \) by \textsc{Find-UB}(\( \mathcal{Y}(L_c) \)). Observe that if \( L_c = \emptyset \) and \( z^u = \textsc{Find-UB}(\mathcal{Y}(L_c)) \) then \( \mathcal{Y} = \mathcal{Y} \cap \{ (y_1, \ldots, y_p) \in \mathbb{R}^p : y_i \leq z^u_i \ \forall i \in \{1, \ldots, p\} \} \).

\textbf{Observation 1.} \( \textsc{Find-UB}(\mathcal{Y}(L_c)) \) requires the solution of \( p \) LPs.

The next propositions and their corollaries provide the basis for the development of our algorithm.

\textbf{Proposition 2.} Let \( z \in \mathbb{R}^p_\leq \) be a point in the payoff space. The intersection of \( L(z) \) and \( H(z) \) is \((\frac{z_1}{p}, \ldots, \frac{z_p}{p})\).

\textbf{Proof.} It is easy to verify that \((\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \in L(z) \) and \((\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \in H(z)\). \hfill \Box

\textbf{Proposition 3.} Let \( z \in \mathbb{R}^p_\leq \) be a point in the payoff space. If \( \mathcal{Y} = \{ y \in \mathbb{R}^p_\geq : \sum_{i=1}^{p} \frac{y_i}{z_i} - 1 \leq 0 \} \) then the optimal point is \( y^* = (\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \).

\textbf{Proof.} To compute the optimal point, the following problem must be solved

\[ y^* \in \arg \max \{ \prod_{i=1}^{p} y_i : y \in \mathcal{Y} \}, \]

which can be rewritten as the following convex optimization problem

\[ y^* \in \arg \min \{ -\sum_{i=1}^{p} \log(y_i) : y \in \mathcal{Y} \}. \]

Relax the problem by assuming that \( y \in \mathbb{R}^p \) (rather than \( y \in \mathbb{R}^p_\geq \)) to obtain,

\[ y_R^* \in \arg \min \{ -\sum_{i=1}^{p} \log(y_i) : \sum_{i=1}^{p} \frac{y_i}{z_i} - 1 \leq 0 \}. \]

The KKT conditions of the relaxed problem are as follows:

\[ -\frac{1}{y_i} + \frac{\eta}{z_i} = 0 \quad \forall i \in \{1, \ldots, p\} \]
\[ \eta(\sum_{i=1}^{p} \frac{y_i}{z_i} - 1) = 0 \]
\[ \sum_{i=1}^{p} \frac{y_i}{z_i} - 1 \leq 0 \]
\[ \eta \geq 0 \]

where \( \eta \) is the dual variable associated with constraint \( \sum_{i=1}^{p} \frac{y_i}{z_i} - 1 \leq 0 \). It is not hard to see that \( y_R^* = (\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \) and \( \eta = p \) are feasible for the KKT conditions. Moreover, \((\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \in \mathbb{R}^p \). Consequently, \( y^* = (\frac{z_1}{p}, \ldots, \frac{z_p}{p}) \). \hfill \Box

\textbf{Corollary 4.} Let \( z, \hat{z} \in \mathbb{R}^p_\leq \) be two points in the payoff space. Let \( y^* \) be the optimal point of a PMP-A with \( \mathcal{Y} = \{ y \in \mathbb{R}^p_\geq : \sum_{i=1}^{p} \frac{y_i}{z_i} - 1 \leq 0 \} \). Let \( \hat{y}^* \) be the optimal point of a PMP-A with \( \hat{\mathcal{Y}} = \{ y \in \mathbb{R}^p_\geq : \sum_{i=1}^{p} \frac{y_i}{\hat{z}_i} - 1 \leq 0 \} \). If the hyperplanes \( \sum_{i=1}^{p} \frac{y_i}{z_i} = 1 \) and \( \sum_{i=1}^{p} \frac{y_i}{\hat{z}_i} = 1 \) are parallel then \( \hat{z}, \hat{y}^*, z, y^* \in L(z) \).
Proof. $z, y^* \in L(z)$ follows immediately from Proposition 3. Moreover, if $\hat{z} \in L(z)$ then $\hat{y}^* \in L(z)$. Consequently, in the rest of the proof, we only show that $\hat{z} \in L(z)$.

Given that the hyperplanes $\sum_{i=1}^{p} \frac{y_i}{z_{i1}} = 1$ and $\sum_{i=1}^{p} \frac{y_i}{z_{i2}} = 1$ are parallel, their normals, i.e., vectors $(\frac{1}{z_{11}}, \ldots, \frac{1}{z_{p1}})$ and $(\frac{1}{z_{12}}, \ldots, \frac{1}{z_{p2}})$, must be parallel too. This implies that $(\frac{1}{z_{11}}, \ldots, \frac{1}{z_{p1}}) = t(\frac{1}{z_{12}}, \ldots, \frac{1}{z_{p2}})$ for some $t \in \mathbb{R}$. Therefore, $z_i = \frac{\hat{z}}{t}$ for all $i \in \{1, \ldots, p\}$, which implies that $\hat{z} \in \{(y_1, \ldots, y_p) \in \mathbb{R}^p : \frac{y_1}{z_{11}} = \frac{y_2}{z_{12}} = \ldots = \frac{y_p}{z_{p1}}\}$.

\begin{proposition}
Let $z \in \mathbb{R}^p_\geq$ be a point in the payoff space. Let $y^*$ be the optimal point of a PMP-A with $\mathcal{Y} = \{y \in \mathbb{R}^p_\geq : \sum_{i=1}^{p} \frac{y_i}{z_{i1}} - 1 \leq 0\}$. Let $\hat{y}^*$ be the optimal point of a PMP-A with $\hat{\mathcal{Y}}$. If $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ and $y^* \in \hat{\mathcal{Y}}$ then $\hat{y}^* = y^*$.
\end{proposition}

\begin{proof}
Since $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$, the PMP-A with $\mathcal{Y}$ is a relaxation of the PMP-A with $\hat{\mathcal{Y}}$.
\end{proof}

\begin{corollary}
Let $z \in \mathbb{R}^p_\geq$ be a point in the payoff space. Let $y^*$ be the optimal point of a PMP-A with $\mathcal{Y} = \{y \in \mathbb{R}^p_\geq : \sum_{i=1}^{p} \frac{y_i}{z_{i1}} - 1 \leq 0\}$. Let $\hat{y}^*$ be the optimal point of a PMP-A with $\hat{\mathcal{Y}}$. If $\hat{\mathcal{Y}} \not\subseteq \mathcal{Y}$ and $y^* \in \hat{\mathcal{Y}}$ then $\hat{y}^* \in (\mathcal{Y} \setminus \hat{\mathcal{Y}}) \cup y^*$.
\end{corollary}

\begin{proof}
The results follows immediately from Propositions 3 and 5.
\end{proof}

Next, we present the details of the algorithm. The algorithm maintains a list of cutting planes $L_c$ and three points $z^l \in \mathcal{Y}$ and $z^u, z^{last} \in \mathbb{R}^p_\geq$. Point $z^l$ is a feasible point and $v(B(z^l))$ gives the best known lower bound for the optimal objective value of the PMP-A. Point $z^u$ is not necessarily a feasible point but $v(B(z^u))$ gives the best known upper bound for the objective value of the PMP-A. Point $z^{last}$ is the upper bound point of the previous iteration. The algorithm terminates when $\frac{v(B(z^u)) - v(B(z^l))}{v(B(z^u))} \leq \epsilon$ for some $\epsilon \geq 0$ specified as part of the input.

Initially, $L_c = \emptyset$, $z^l = (0, \ldots, 0)$ and $z^u = \text{FIND-UB}(\mathcal{Y}(\emptyset))$. Next, we describe how the values of $z^l$ and $z^u$ are computed in each iteration.

- **Determining $z^l$:** In each iteration, we use $\text{FIND-INTERSECTION}(z^u)$ to compute the intersection of $z^l$ of $L(z^u)$ with the common boundaries of $\mathcal{Y}(L_c)$ and $\mathcal{Y}$. If $v(B(z^l)) > v(B(z^l))$, we set $z^l$ equal to $z^l$. An illustration for the case when $p = 2$ can be found in Figure 1.

![Figure 1: The first upper bound and intersection points.](image-url)
• Determining $z^u$: In each iteration, after determining $z^l$, we update $z^u$. Corollary 4 guarantees that $z^l$ is the optimal point for a PMP-A with a feasible set $Y$ defined by a single hyperplane that is parallel to $H(z^u)$ and contains $z^l$. If $\mathcal{Y}(L_c) \subseteq \bar{\mathcal{Y}}$, then, by Proposition 5 and Corollary 6, $z^l$ is the optimal point for $\mathcal{Y}(L_c)$. Furthermore, if $\mathcal{Y}(L_c) \not\subseteq \bar{\mathcal{Y}}$, then we only need to search $\mathcal{Y}(L_c) \setminus \bar{\mathcal{Y}}$ to find a better solution.

Therefore, in each iteration, we add the constraint $C(z^u, z^l)$ to $L_c$, and call FIND-UB($\mathcal{Y}(L_c)$) to update $z^u$. If $z^{last} = z^u$, we set $z^u$ to $z^l$, which is the optimal point in this situation and no further searching is required (see also Observation 8). Otherwise, we set $z^{last}$ to $z^u$. An illustration of determining $z^u$ for the case $p = 2$ can be found in Figures 2 and 3.

![Figure 2: The cutting plane $C(z^u, z^l)$](image1.png)  
![Figure 3: The new upper bound and intersection points](image2.png)

A precise description of the algorithm can be found in Algorithm 1. Next, we make a number of observations about the algorithm and show that to achieve a relative gap of $\epsilon$, the algorithm requires a number of iterations that is bounded by a quadratic in $p$. For convenience, let $z^u$, $z^l$, $\hat{z}^u$, and $\hat{z}^l$ be the upper bounds and intersection points in two consecutive iterations.

**Observation 7.** By construction, the new upper bound point $\hat{z}^u$ has $z^l_i \leq \hat{z}^u_i \leq z^u_i$ for all $i \in \{1, \ldots, p\}$ and $\hat{z}^u_j = z^l_j$ for at least one $j \in \{1, \ldots, p\}$.

**Observation 8.** If $\hat{z}^u = z^u$, then $\mathcal{Y}(L_c) \subseteq \{y \in \mathbb{R}^p : \sum_{i=1}^p \frac{y_i}{\hat{z}^u_i} \leq \sum_{i=1}^p \frac{z^u_i}{\hat{z}^u_i} \}$ and $z^l$ is the optimal point for set $\mathcal{Y}(L_c)$. (Note that $\sum_{i=1}^p \frac{y_i}{\hat{z}^u_i} = \sum_{i=1}^p \frac{z^l_i}{\hat{z}^u_i}$ is essentially $H(z^u)$ shifted so that it contains $z^l$.)

**Proposition 9.** Let $z^u$, $z^l$, $\hat{z}^u$, and $\hat{z}^l$ be the upper bounds and intersection points in two consecutive iterations. Then $d(\hat{z}^u, \hat{z}^l) \leq \frac{\epsilon^2 - \epsilon}{p^2 - p + 1} d(z^u, z^l)$.

**Proof.** Let $\hat{z}^l$ be the intersection of $L(\hat{z}^u)$ and $H(\hat{z}^u)$ shifted so that it contains $z^l$. An illustration of the situation for the case $p = 2$ can be found in Figure 4. Due to the structure of the algorithm and the convexity of the set $\mathcal{Y}(L_c)$, we have $d(\hat{z}^u, \hat{z}^l) \leq d(z^u, z^l)$. Therefore, it suffices to show that $d(\hat{z}^u, \hat{z}^l) \leq \frac{\epsilon^2 - \epsilon}{p^2 - p + 1} d(z^u, z^l)$.

To prove $d(\hat{z}^u, \hat{z}^l) \leq \frac{\epsilon^2 - \epsilon}{p^2 - p + 1} d(z^u, z^l)$, we consider the worst case scenario, i.e., when $d(\hat{z}^u, \hat{z}^l)$ is as large as possible. By construction, we have that $\hat{z}^u_i \leq z^u_i$ for all $i \in \{1, \ldots, p\}$, but $\hat{z}^u \neq z^u$ (Observation 7). Next, we will now show that $d(\hat{z}^u, \hat{z}^l)$ is as large as possible when $\hat{z}^u$ and $z^u$ differ in exactly one component.
Algorithm 1:

Input: A PMP-A

$L_c \leftarrow \emptyset$

$z^l \leftarrow (0, \ldots, 0)$

$z^u \leftarrow \text{Find-UB} (Y(L_c))$

$z^{last} \leftarrow z^u$

while $\frac{v(B(z^u)) - v(B(z^l))}{v(B(z^u))} > \epsilon$ do

$z^l \leftarrow \text{Find-Intersection}(z^u)$

if $v(B(z^l)) > v(B(z^u))$ then

$z^l \leftarrow z^l$

if $\frac{v(B(z^u)) - v(B(z^l))}{v(B(z^u))} > \epsilon$ then

Add $C(z^u, z^l)$ to $L_c$

$z^u \leftarrow \text{Find-UB} (Y(L_c))$

if $z^u = z^{last}$ then

$z^u \leftarrow z^l$

else

$z^{last} \leftarrow z^u$

return $z^l$

Figure 4: An illustration for proof of Proposition 9 for the case $p = 2$. 
Let $H(z^u,z^l)$ denote the hyperplane $H(z^u)$ shifted so that it contains $z^l$. $H(z^u,z^l)$ can be represented as follows

$$\{y \in \mathbb{R}^p : \sum_{i=1}^{p} y_i \frac{z_i^u}{z_i^l} = \sum_{i=1}^{p} z_i^l \}.$$ 

Let $L(\hat{z}^u) = t\hat{z}^u$, for some $t \in \mathbb{R}$, be the vector representation of line $L(z^u)$. Because $\hat{z}^l \in L(\hat{z}^u)$, and $\hat{z}^l \in H(z^u,z^l)$, we have

$$\sum_{i=1}^{p} \frac{t\hat{z}_i^u}{z_i^l} = \sum_{i=1}^{p} \frac{z_i^l}{z_i^l} = t = \frac{\sum_{i=1}^{p} z_i^l}{\sum_{i=1}^{p} z_i^l}.$$ 

Because $\hat{z}^u$ has $z_i^l \leq \hat{z}_i^u \leq z_i^u$ for all $i \in \{1, \ldots, p\}$ and $\hat{z}_j^u = z_j^l$ for at least one $j \in \{1, \ldots, p\}$ (Observation 7) and $\hat{z}^l \in L(\hat{z}^u)$, we have

$$t \geq \frac{p z_j^l}{(p-1) \frac{z_i^u}{z_i^l} + \frac{z_i^l}{z_j^l}}.$$ 

If, for notational convenience, we let $x = \frac{z_j^l}{z_j^l}$, then

$$t \geq \frac{px}{(p-1) + x} \Rightarrow \hat{z}^l \geq \frac{px}{(p-1) + x} \hat{z}^u.$$ 

Consequently, because $0 < x < 1$, $d(\hat{z}^u, \hat{z}^l)$ is as large as possible when $\hat{z}^u$ and $z^u$ differ in exactly one component (i.e., when $\hat{z}^l = \frac{px}{(p-1)+x} \hat{z}^u$).

For the remainder, therefore, we assume that $\hat{z}_i^u = z_i^u$ for $i \in \{1, \ldots, p\} \setminus \{j\}$ and $\hat{z}_j^u \neq z_j^u$. To prove that $d(\hat{z}^u, \hat{z}^l) \leq \frac{p^2-p}{p^2-p+1} d(z^u, z^l)$, we show that $\hat{z}_i^u - \hat{z}_i^l \leq \frac{p^2-p}{p^2-p+1} (z_i^u - z_i^l)$ for all $i \in \{1, \ldots, p\}$.

**Part 1:** $\hat{z}_i^u - \hat{z}_i^l \leq \frac{p^2-p}{p^2-p+1} (z_i^u - z_i^l)$ for all $i \in \{1, \ldots, p\} \setminus \{j\}$

For all $i \in \{1, \ldots, p\} \setminus \{j\}$, there must exists a $0 \leq \lambda_i \leq 1$ such that

$$\hat{z}_i^u - \hat{z}_i^l = \lambda_i (z_i^u - z_i^l).$$ 

Because $\hat{z}_i^u = z_i^u$, this is equivalent to

$$z_i^u (1 - \frac{px}{(p-1)+x}) = \lambda_i (z_i^u - z_i^l),$$

which can be simplified to,

$$1 - \frac{px}{(p-1)+x} = \lambda_i (1 - \frac{z_i^l}{z_i^u}).$$

Because $z^l \in L(z^u)$, we have $\frac{z_i^l}{z_i^u} = \frac{z_i^l}{z_i^l} = x$, and thus,

$$\lambda_i = \frac{1 - \frac{px}{(p-1)+x}}{(1-x)}.$$ 

To maximize $\hat{z}_i^u - \hat{z}_i^l$, we need to find

$$\lambda_i^* = \max_{0 < x < 1} \left(1 - \frac{px}{(p-1)+x}\right) \frac{1}{(1-x)}.$$ 

10
By Proposition 2, the intersection of $H(z^u)$ and $L(z^u)$ is $(\frac{z^u_i}{p}, \ldots, \frac{z^u_i}{p})$. As a consequence, if $x < \frac{1}{p}$, then $z^u_i < \frac{z^u_i}{p}$ for all $i \in \{1, \ldots, p\}$, which would imply that the set $\mathcal{Y}(L_c)$ is not convex; a contradiction. Now, it is not hard to see that the maximum is obtained by setting that $x = \frac{1}{p}$, i.e.,

$$\lambda^*_j = \frac{(1 - \frac{p^2}{(p-1)+\frac{p}{2}})}{(1 - \frac{1}{p})}.$$  

**Part 2:** $\hat{z}^u_j - \hat{z}^l_j \leq \frac{p^2 - p}{p^2 - p + 1}(z^u_j - z^l_j)$

Our analysis proceeds similar to Part 1. There must exists a $0 \leq \lambda_j \leq 1$ such that

$$\hat{z}^u_j - \hat{z}^l_j = \lambda_j(z^u_j - z^l_j) \Rightarrow \hat{z}^u_j(1 - \frac{px}{(p-1) + x}) = \lambda_j(z^u_j - z^l_j).$$

Recalling that if $z^u_j \neq z^l_j$, we have $\hat{z}^u_j = z^l_j$, this implies

$$z^l_j(1 - \frac{px}{(p-1) + x}) = \lambda_j(z^u_j - z^l_j) \Rightarrow (1 - \frac{px}{(p-1) + x}) = \lambda_j(\frac{1}{x} - 1).$$

To maximize $\hat{z}^u_j - \hat{z}^l_j$, we need to find

$$\lambda^*_j = \max_{0 < x < 1} \left( 1 - \frac{px}{(p-1) + x} \right) \left( \frac{1}{x} - 1 \right).$$

It is not hard to see that the maximum is obtained when $x \to 1$, i.e.,

$$\lambda^*_j = \lim_{x \to 1} \left( 1 - \frac{px}{(p-1) + x} \right) = \frac{p-1}{p}.$$  

Since $p \geq 1$

$$\lambda^*_j \leq \frac{p^2 - p}{p^2 - p + 1}. \quad \square$$

**Theorem 10.** The algorithm finds the optimal point of a PMP-A in polynomial time, for any given relative gap $\epsilon > 0$.

**Proof.** In each iteration, the algorithm solves $p + 1$ (compact) linear programs. So, we only need to show that the number of iterations is bounded by a polynomial. Proposition 9 shows that the distance between the upper and lower bound points improves at least by a factor $\frac{p^2 - p}{p^2 - p + 1}$ in each iteration. Observe that there must exist an $\epsilon' > 0$ such that $d(z^u, z^l) \leq \epsilon'$ is equivalent to $\frac{v(B(z^u)) - v(B(z^l))}{v(B(z^u))} \leq \epsilon$. Let $d^0$ be the initial distance between the upper and lower bound points in the algorithm. As a consequence, the number of required iterations $\beta$ to achieve a relative gap of at most $\epsilon$ is

$$\beta \leq \log \frac{x^2 - x}{p^2 - p + 1} \frac{\epsilon'}{d^0},$$

which can be rewritten as

$$\beta \leq \frac{\log \frac{\epsilon'}{p^2}}{\log \frac{p^2 - p}{p^2 - p + 1}} = -\frac{\log \frac{d^0}{p^2}}{-\log(1 + \frac{1}{p^2 - p})} = \frac{\log \frac{d^0}{p^2}}{\log(1 + \frac{1}{p^2 - p})}.$$  

Using a Taylor series to approximate the log function, it is not hard to see that

$$\beta \leq O(p^2). \quad \square$$
Corollary 11. For a given relative gap \( \epsilon > 0 \), the number of iterations of the algorithm is bounded by \( O(p^2) \).

Corollary 12. For a given relative gap \( \epsilon > 0 \), the algorithm solves \( O(p^3) \) LPs to obtain the optimal point of a PMP-A.

We conclude this section with some comments about the algorithm.

- The set \( \mathcal{Y}(L_c) \) is usually much smaller than the set \( \mathcal{Y} \). Therefore, numerical issues may arise in \textsc{Find-Intersection}(z) (because we are searching along a line). To reduce the likelihood of numerical issues, we use \( \mathcal{Y} \) rather than \( \mathcal{Y}(L_c) \) in \textsc{Find-Intersection}(z), i.e.,

\[
z^I = \arg \max \left\{ \sum_{i=1}^{p} y_i : y \in \mathcal{Y}, y \in L(z) \right\}.
\]

- It is relatively easy to construct different variants of the algorithm. For instance by developing different strategies for computing \( z^u \) in each iteration. We believe the variant that we have described is the most natural one. We now briefly illustrate another variant of the algorithm, which may be useful for some cases. The main difference is that we start each iteration with \( L_c = \emptyset \). Moreover, in each iteration when we call \textsc{Find-UB}(\mathcal{Y}(L_c)), we use \( z^u_i = \max_{y \in \mathcal{Y}(L_c) : y \leq z^{\text{last}}} y_i \) for all \( i \in \{1, \ldots, p\} \) (rather than \( z^u_i = \max_{y \in \mathcal{Y}(L_c)} y_i \)). The main advantage of using this variant is that \( L_c \) contains at most one cutting plane at any time.

- In some applications of PMP-As, e.g., nonsymmetric bargaining games, some of the variables in the objective function have multiplicity greater than one. In Section 2, we discussed that in these cases, copies of those variables can be introduced to change the objective function to the standard multi-linear objective function. It is evident that when using the operation \textsc{Find-UB}(\mathcal{Y}(L_c)) during the course of algorithm, we do not have to solve any linear programs for copies of a same variable (because by definition the value of each of copies are equal to the original value). As a consequence, for such a problem, the efficiency of the algorithm does not change.

4 A Computational Study

In this section, we compare the performance of our proposed algorithm with that of IPOPT solver 3.10.2 (called from AMPL). Our algorithm is implemented in C++ and uses CPLEX 12.6 as the LP solver. The computational experiments were conducted on a Dell PowerEdge R710 with dual hex core 3.06Gz Intel Xeon X5675 processors and 96GB RAM, with the RedHat Enterprise Linux 6 operating system, and using a single thread. We also set the relative optimality gap of CPLEX and our algorithm to \( 10^{-12} \). We noticed that IPOPT performs better when the logarithmic transformation is applied to the objective function of a PMP-A, so this is done for all instances.

In order to test the performance of our algorithm, a total of 180 problems were randomly generated. The sparsity of matrix \( A \) was set to 75\%. The components of vector \( b \) and entries of matrix \( A \) were randomly drawn from discrete uniform distributions \([50, 200]\) and \([1, 30]\), respectively. We set the components of vector \( d \) to zero. The sparsity of each row of the matrix \( D \) was also set to 75\% and its components were drawn randomly from a discrete uniform distribution \([1, 10]\). Note that since all constraints of the set \( \mathcal{X} \) are inequality constraints and all coefficients of matrix \( A \) are nonnegative, the set \( \mathcal{X} \) is bounded.

There are three sets of 60 problems with \( p = 2, 4, \) and 6, respectively. Each set has 12 subclasses depending on the dimensions of matrix \( A_{m \times n} \) including, \( 500 \times 625, 500 \times 750, 500 \times 875, 500 \times 1000, 1000 \times 1250, 1000 \times 1500, 1000 \times 1750, 1000 \times 2000, 2000 \times 2500, 2000 \times 3000, 2000 \times 3500, 2000 \times 4000, \) and each subclass contains 5 problems.
We imposed a run time limitation of 900 seconds for each problem. Detailed performance statistics of our algorithm and IPOPT can be found in tables 1 to 3, where we report the number iterations (#Iters), the number of LPs (#LP), and the time taken to solve the problem (Time(sec.)). In the result tables, averages over five instances are reported.

Observe that our algorithm is able to solve all problems within the imposed time limit; even the largest size problems are solved within one minute. IPOPT is much slower than our algorithm and cannot solve the largest size problems with the imposed time limit. Moreover, the results show that our algorithm’s runtime grows much slower than IPOPT’s runtime as $n_1$ and $n_2$ increase. The effect of $p$ on the runtime of our algorithm is less clear, because the number of iterations of our algorithm does not necessary increase as $p$ grows. In general, it seems that IPOPT’s runtime is more robust than our algorithm’s runtime with respect to $p$, meaning that as $p$ increases its runtime increases smoothly without significant jumps.

### Table 1: Results for class $p = 2$

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Algorithm 1</th>
<th>IPOPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Iers</td>
<td>#LPs</td>
</tr>
<tr>
<td>500 $\times$ 625</td>
<td>13.8</td>
<td>43.4</td>
</tr>
<tr>
<td>500 $\times$ 750</td>
<td>11.0</td>
<td>35.0</td>
</tr>
<tr>
<td>500 $\times$ 875</td>
<td>11.0</td>
<td>35.0</td>
</tr>
<tr>
<td>500 $\times$ 1000</td>
<td>11.0</td>
<td>35.0</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>11.70</strong></td>
<td><strong>37.10</strong></td>
</tr>
<tr>
<td>1000 $\times$ 1250</td>
<td>10.8</td>
<td>34.4</td>
</tr>
<tr>
<td>1000 $\times$ 1500</td>
<td>12.4</td>
<td>39.2</td>
</tr>
<tr>
<td>1000 $\times$ 1750</td>
<td>11.6</td>
<td>36.8</td>
</tr>
<tr>
<td>1000 $\times$ 2000</td>
<td>16.2</td>
<td>50.6</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>12.75</strong></td>
<td><strong>40.25</strong></td>
</tr>
<tr>
<td>2000 $\times$ 2500</td>
<td>11.8</td>
<td>37.4</td>
</tr>
<tr>
<td>2000 $\times$ 3000</td>
<td>9.2</td>
<td>29.6</td>
</tr>
<tr>
<td>2000 $\times$ 3500</td>
<td>10.0</td>
<td>32.0</td>
</tr>
<tr>
<td>2000 $\times$ 4000</td>
<td>9.6</td>
<td>30.8</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>10.15</strong></td>
<td><strong>32.45</strong></td>
</tr>
</tbody>
</table>

### Table 2: Results for class $p = 4$

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Algorithm 1</th>
<th>IPOPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Iers</td>
<td>#LPs</td>
</tr>
<tr>
<td>500 $\times$ 625</td>
<td>26.0</td>
<td>134.0</td>
</tr>
<tr>
<td>500 $\times$ 750</td>
<td>16.6</td>
<td>87.0</td>
</tr>
<tr>
<td>500 $\times$ 875</td>
<td>12.0</td>
<td>64.0</td>
</tr>
<tr>
<td>500 $\times$ 1000</td>
<td>13.0</td>
<td>69.0</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>16.90</strong></td>
<td><strong>88.50</strong></td>
</tr>
<tr>
<td>1000 $\times$ 1250</td>
<td>11.2</td>
<td>60.0</td>
</tr>
<tr>
<td>1000 $\times$ 1500</td>
<td>12.0</td>
<td>64.0</td>
</tr>
<tr>
<td>1000 $\times$ 1750</td>
<td>30.6</td>
<td>157.0</td>
</tr>
<tr>
<td>1000 $\times$ 2000</td>
<td>13.4</td>
<td>71.0</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>16.80</strong></td>
<td><strong>88.00</strong></td>
</tr>
<tr>
<td>2000 $\times$ 2500</td>
<td>21.4</td>
<td>111.0</td>
</tr>
<tr>
<td>2000 $\times$ 3000</td>
<td>19.4</td>
<td>101.0</td>
</tr>
<tr>
<td>2000 $\times$ 3500</td>
<td>15.4</td>
<td>81.0</td>
</tr>
<tr>
<td>2000 $\times$ 4000</td>
<td>11.8</td>
<td>63.0</td>
</tr>
<tr>
<td>Avg</td>
<td><strong>17.00</strong></td>
<td><strong>89.00</strong></td>
</tr>
</tbody>
</table>

### 5 Some Potential Applications in Other Fields

We now discuss some potential applications of PMP-As in other fields of study. We refer the interested readers to Boyd and Vandenberghe [1] for further information.
### Table 3: Results for class $p = 6$

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Algorithm 1</th>
<th></th>
<th>IPOPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Iters</td>
<td>#LPs</td>
<td>Time (sec.)</td>
</tr>
<tr>
<td>500 × 625</td>
<td>9.2</td>
<td>70.4</td>
<td>3.2</td>
</tr>
<tr>
<td>500 × 750</td>
<td>8.8</td>
<td>67.6</td>
<td>3.8</td>
</tr>
<tr>
<td>500 × 875</td>
<td>13.6</td>
<td>101.2</td>
<td>7.3</td>
</tr>
<tr>
<td>500 × 1000</td>
<td>8.0</td>
<td>62.0</td>
<td>4.7</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td>9.90</td>
<td>75.30</td>
<td>4.72</td>
</tr>
<tr>
<td>1000 × 1250</td>
<td>11.6</td>
<td>87.2</td>
<td>19.4</td>
</tr>
<tr>
<td>1000 × 1500</td>
<td>13.8</td>
<td>102.6</td>
<td>36.0</td>
</tr>
<tr>
<td>1000 × 1750</td>
<td>11.6</td>
<td>87.2</td>
<td>25.9</td>
</tr>
<tr>
<td>1000 × 2000</td>
<td>8.8</td>
<td>67.6</td>
<td>34.7</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td>11.45</td>
<td>86.15</td>
<td>29.02</td>
</tr>
<tr>
<td>2000 × 2500</td>
<td>12.6</td>
<td>94.2</td>
<td>126.5</td>
</tr>
<tr>
<td>2000 × 3000</td>
<td>13.4</td>
<td>99.8</td>
<td>184.8</td>
</tr>
<tr>
<td>2000 × 3500</td>
<td>11.6</td>
<td>87.2</td>
<td>150.9</td>
</tr>
<tr>
<td>2000 × 4000</td>
<td>14.8</td>
<td>109.6</td>
<td>210.1</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td>13.10</td>
<td>97.70</td>
<td>168.06</td>
</tr>
</tbody>
</table>

#### 5.1 Geometry

Computing the analytic center of a set of linear inequalities

$$a_i^T x \leq b_i \quad \forall i \in \{1, \cdots, p\},$$

which can be done by solving

$$x^* \in \arg \min \{-\sum_{i=1}^{p} \log(b_i - a_i^T x)\}.$$ 

Therefore, this problem can easily be transformed to a PMP-A, i.e.,

$$y^* \in \arg \max \{ \prod_{i=1}^{p} y_i : y \in \mathcal{Y} \},$$

where \( \mathcal{Y} = \{ y \in \mathbb{R}_0^p : y_i = b_i - a_i^T x \ \forall i \in \{1, \cdots, p\} \}. \) Observe that \( y_i \) is the distance to hyperplane \( \{ x \in \mathbb{R}^n : a_i^T x = b_i \} \) and \( x^* \) is the analytical center. (Note that side constraints can be added, e.g., \( x \in \mathcal{X} \), without difficulties.)

#### 5.2 Statistical Estimation

We consider a linear measurement model,

$$b_i = a_i^T x + r_i \quad \forall i \in \{1, \cdots, p\},$$

where \( b \in \mathbb{R}^p \) is a vector of observed quantities, \( r \) is a vector of measurement errors (noise), \( x \in \mathbb{R}^n \) is a vector of parameters to be estimated, and \( a_i^T \in \mathbb{R}^n \) are rows of the matrix of coefficients.

We assume that \( r_i \) are independent, identically distributed (IID), with a density \( z(.) \). If \( z(.) \) is a linear function then the maximum likelihood estimate can be found by solving a PMP-A,

$$y^* \in \arg \max \{ \prod_{i=1}^{p} y_i : y \in \mathcal{Y} \},$$

where \( \mathcal{Y} = \{ y \in \mathbb{R}_0^p : y_i = z(b_i - a_i^T x) \ \forall i \in \{1, \cdots, p\}, \ x \in \mathcal{X} \} \) where \( \mathcal{X} \) provides some side constraints or prior information on the parameter vector.

#### 5.3 Approximation and Fitting

The log barrier regression (approximation) problem can be stated as follows,

$$x^* \in \arg \min \{ \sum_{i=1}^{m} -c_i^2 \log(1 - \frac{r_i^2}{c_i^2}) : r = Ax - b \},$$
where \( x \in \mathbb{R}^n \) is a vector of decision variables, \( r \in \mathbb{R}^m \) is a vector of auxiliary variables, \( A \) is a \( m \times n \) matrix of coefficients, \( b \in \mathbb{R}^m \) is a vector of observed quantities, and \( c_i \) is a positive integer number for all \( i \in \{1, \cdots, m\} \).

The solution of log barrier regression problem \( x^* \) is sometimes called an approximate solution and the vector \( r \) is called residual of the problem. Moreover, columns of matrix \( A \) are usually called regressors.

The goal of log barrier regression problem is to fit or approximate the vector \( b \) by a linear combination of regressors, as closely as possible. The deviation is measured by the log barrier penalty function \( f(v) \),

\[
    f(v) = \begin{cases} 
    -d^2 \log(1 - \frac{v^2}{d^2}) & |v| < d \\
    +\infty & |v| \geq d 
    \end{cases}
\]

where \( d \) is a positive integer number. The log barrier function assigns an infinite penalty to residuals with an absolute value greater than or equal to \( d \).

The log barrier regression problem is equivalent to

\[
    x^* \in \arg \max \left\{ \prod_{i=1}^m (1 - \frac{r_i^2}{c_i^2})^{c_i^2} : r = Ax - b \right\}.
\]

Therefore, it can be simplified to

\[
    x^* \in \arg \max \left\{ \prod_{i=1}^m (1 - \frac{r_i}{c_i})^{c_i} (1 + \frac{r_i}{c_i})^{c_i} : r = Ax - b \right\}.
\]

Observe that \( (1 - \frac{r_i}{c_i}) \) and \( (1 + \frac{r_i}{c_i}) \) are linear. As a consequence, the log barrier regression problem can be transformed to a PMP-A with similar techniques that we described for the bargaining problem.

### 5.4 Polynomial Programming

Any polynomial programming problem with only affine constraints and an objective function that can be factorized into \( p \) linear functions, i.e., \( z_i(x) \) where \( i \in \{1, \ldots, p\} \), can be formulated as a PMP-A if \( z_i(x) \geq 0 \) for all \( x \in X \) and for all \( i \in \{1, \ldots, p\} \). (Mild condition: there exists a feasible solution \( x \in X \) where \( z_i(x) > 0 \) for all \( i \in \{1, \ldots, p\} \).

### 6 Conclusion

We introduced the first polynomial time LP-based method to compute an optimal solution to a relevant class of optimization problems with a multi-linear objective function and affine constraints. The algorithm solves \( O(p^3) \) linear programs.

The algorithm can be applied to optimization problems arising in several fields, including game theory, geometry, statistical estimation, approximation and fitting, and polynomial programming.

Our algorithm can be modified to be used in compact non-convex problems, in which the non-convexity comes from the existence of integer decision variables in the model, through a branch-and-bound algorithm.

### References


