Mixed-Integer Programming Methods for Finding Nash Equilibria*

Tuomas Sandholm and **Andrew Gilpin** and **Vincent Conitzer**
Carnegie Mellon University
Computer Science Department
{sandholm,gilpin,conitzer}@cs.cmu.edu

Abstract

We present, to our knowledge, the first mixed integer program (MIP) formulations for finding Nash equilibria in games (specifically, two-player normal form games). We study different design dimensions of search algorithms that are based on those formulations. Our *MIP Nash* algorithm outperforms *Lemke-Howson* but not *Porter-Nudelman-Shoham (PNS)* on GAMUT data. We argue why experiments should also be conducted on games with equilibria with medium-sized supports only, and present a methodology for generating such games. On such games *MIP Nash* drastically outperforms *PNS* but not *Lemke-Howson*. Certain *MIP Nash* formulations also yield anytime algorithms for ϵ-equilibrium, with provable bounds. Another advantage of *MIP Nash* is that it can be used to find an *optimal* equilibrium (according to various objectives). The prior algorithms can be extended to that setting, but they are orders of magnitude slower.

Introduction

Nash equilibrium (Nash 1950) is the most central solution concept for games. It defines how rational agents should act in settings where an agent's best strategy may depend on what another agent does, and vice versa. For the concept to be operational, it needs to be accompanied by an algorithm for finding an equilibrium. While the concept was invented in 1950, it remains unknown whether an equilibrium can be found in polynomial time, even in 2-agent games.

In a 2-agent normal form game, the focus of this paper, each agent i has a finite set S_i of pure strategies to choose from, and the agent's utility is $u_i(s_i, s_{1-i})$, where $s_i \in S_i$ is the agent's chosen pure strategy, and s_{1-i} is the other agent's chosen pure strategy. Each agent i can also use a *mixed strategy*, i.e., randomize over the pure strategies in S_i (according to probabilities p_{s_i} which sum to 1). The pure strategies that an agent plays with nonzero probability are called that agent's *support*. The (mixed) strategies p_{s_i} are in *Nash equilibrium* if neither agent has an incentive to alter his probabilities given that the other does not alter hers: for both agents $i \in \{0, 1\}$, for any mixed strategy p'_{s_i}, $\sum_{s_i \in S_i} p_{s_i} \sum_{s_{1-i} \in S_{1-i}} p_{s_{1-i}} u_i(s_i, s_{1-i}) \geq \sum_{s_i \in S_i} p'_{s_i} \sum_{s_{1-i} \in S_{1-i}} p_{s_{1-i}} u_i(s_i, s_{1-i})$. At least one equilibrium exists in any such game (Nash 1950).

The question of how complex it is to construct a Nash equilibrium has been dubbed "a most fundamental computational problem whose complexity is wide open" and "together with factoring, [...] the most important concrete open question on the boundary of P today" (Papadimitriou 2001). Until recently, the *Lemke-Howson* algorithm (Lemke & Howson 1964) was the most efficient method for finding an equilibrium in a 2-agent game.[1] It is a path-following method that finds an equilibrium by pivoting through complementary feasible bases for the corresponding linear complementarity problem. It takes exponentially many steps in the worst case (Savani & von Stengel 2004).

A recent paper describes a simple search method, which we refer to as *PNS*, for finding Nash equilibria (Porter, Nudelman, & Shoham 2004). It enumerates strategy supports and determines whether the support yields a feasible solution to the equilibrium problem. Although that idea has been previously described (e.g. (Dickhaut & Kaplan 1991), (Myerson 1991, Section 3.3)), the paper improved on the basic idea by adding dominance checks and a well-motivated search bias, and reported the first computational experience with that approach. In the experiments, *PNS* was significantly faster than *Lemke-Howson*.

We present search algorithms based on *mixed integer program* formulations of the Nash equilibrium finding problem. (A mixed integer program is a linear program in which some of the variables are constrained to be integers.) We develop algorithms for finding a Nash equilibrium, anytime algorithms for finding an approximate equilibrium, and algorithms for finding an optimal Nash equilibrium—according to a variety of criteria. We also provide experimental validation of this approach using a modern MIP solver. For many (but not all) problems the new algorithms outperform the prior state of the art.

Mixed-integer program (MIP) formulations

The *regret* of pure strategy s_i is the difference in utility for player i between playing an optimal strategy (given the other

*This material is based upon work supported by the National Science Foundation under ITR grants IIS-0121678 and IIS-0427858, and a Sloan Fellowship.

[1] Under various convexity assumptions, Nash equilibria can be found using continuous optimization (Antipin 2003; Khamisov 2003).

player's mixed strategy) and playing s_i. Our mixed integer programs are based on the following simple observation (which could be said to underlie the PNS algorithm as well): *In any equilibrium, every pure strategy is either played with probability* 0*, or has* 0 *regret. Also, any vector of mixed strategies for the players where every pure strategy is either played with probability* 0*, or has* 0 *regret, is an equilibrium.*

Based on this observation, we introduce four MIP formulations for finding an equilibrium. In the first, the equilibria are the only feasible solutions. Therefore, this formulation allows us to specify an objective to be optimized over the space of equilibria. For instance, we can find a social-welfare maximizing equilibrium. The other three formulations have feasible solutions other than the equilibria as well; the equilibria are exactly the solutions that minimize the objective. The benefit of this latter approach is that even when the solver has not yet found an equilibrium, it may have already found something that is close (in a precise sense, discussed later). This also yields a measure of progress towards finding an equilibrium. On the other hand, using the MIP objective in defining equilibrium makes it more difficult to use these formulations to optimize an objective over the space of equilibria. (We discuss how this can nevertheless be done.)

Formulation 1: Only equilibria are feasible

In our first formulation, the feasible solutions are exactly the equilibria of the game. For every pure strategy s_i, there is a binary variable b_{s_i}. If this variable is set to 1, the probability placed on the strategy must be 0. If it is set to 0, the strategy is allowed to be in the support, but the regret of the strategy must be 0. The formulation has the following variables other than the b_{s_i}. For each player, there is a variable u_i indicating the highest possible expected utility that that player can obtain given the other player's mixed strategy. For every pure strategy s_i, there is a variable p_{s_i} indicating the probability placed on that strategy, a variable u_{s_i} indicating the expected utility of playing that strategy (given the other player's mixed strategy), and a variable r_{s_i} indicating the regret of playing s_i. The constant U_i indicates the maximum difference between two utilities in the game for player i: $U_i = \max_{s_i^h, s_i^l \in S_i, s_{1-i}^h, s_{1-i}^l \in S_{1-i}} u_i(s_i^h, s_{1-i}^h) - u_i(s_i^l, s_{1-i}^l)$. The formulation follows below.

find $p_{s_i}, u_i, u_{s_i}, r_{s_i}, b_{s_i}$ **such that**

$$(\forall i) \sum_{s_i \in S_i} p_{s_i} = 1 \quad (1)$$

$$(\forall i)(\forall s_i \in S_i) \quad u_{s_i} = \sum_{s_{1-i} \in S_{1-i}} p_{s_{1-i}} u_i(s_i, s_{1-i}) \quad (2)$$

$$(\forall i)(\forall s_i \in S_i) \quad u_i \geq u_{s_i} \quad (3)$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} = u_i - u_{s_i} \quad (4)$$

$$(\forall i)(\forall s_i \in S_i) \quad p_{s_i} \leq 1 - b_{s_i} \quad (5)$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} \leq U_i b_{s_i} \quad (6)$$

domains: $p_{s_i} \geq 0, u_i \geq 0, u_{s_i} \geq 0, r_{s_i} \geq 0, b_{s_i} \in \{0, 1\}$.

The first four constraints ensure that the p_{s_i} values constitute a valid probability distribution and define the regret of a strategy. Constraint 5 ensures that b_{s_i} can be set to 1 only when no probability is placed on s_i. On the other hand, Constraint 6 ensures that the regret of a strategy equals 0, unless $b_{s_i} = 1$, in which case the constraint is vacuous because the regret can never exceed U_i. (Technically, Constraint 3 is redundant as it follows from Constraint 4 and $r_{s_i} \geq 0$.)

Formulation 2: Penalize regret on strategies that are played with positive probability

The formulation in this subsection allows for feasible solutions in which pure strategies that are played with positive probability have positive regret. However, this regret is counted as a penalty in the objective. Strategies that have no probability placed on them are exempt from having this penalty counted, which is done using binary variables b_{s_i} that can be set to 1 if and only if no probability is placed on the corresponding strategy s_i. The variables in the formulation are the same as in the first formulation, with the addition of a variable f_{s_i} for every $s_i \in S_i$. The formulation is:

minimize $\sum_{i=0}^{1} \sum_{s_i \in S_i} f_{s_i} - U_i b_{s_i}$ **subject to**

Constraints 1, 2, 3, 4, 5, the domains from Formulation 1, and

$$(\forall i)(\forall s_i \in S_i) \quad f_{s_i} \geq r_{s_i} \quad (7)$$

$$(\forall i)(\forall s_i \in S_i) \quad f_{s_i} \geq U_i b_{s_i} \quad (8)$$

When b_{s_i} is set to 1 in this formulation (which can be done only when the probability on that strategy is 0, by Constraint 5), f_{s_i} must be set to U_i, which then cancels out with the $-U_i b_{s_i}$ term in the objective. However, when b_{s_i} is set to 0, f_{s_i} must be set to r_{s_i} (and the $-U_i b_{s_i}$ term in the objective will equal 0). Thus the objective is indeed to minimize the sum of the regrets of strategies that have positive probability.

Formulation 3: Penalize probability placed on strategies with positive regret

The formulation in this subsection is similar to the previous one; the difference is that instead of counting the regret on strategies that are played with positive probability as a penalty, we count the probability that is placed on strategies that have positive regret as a penalty. Again, this is done through the use of binary variables b_{s_i} that can be set to 0 if and only if the corresponding strategy s_i has no regret. Again, the variables in the formulation are the same as in Formulation 1, with the addition of g_{s_i} for every $s_i \in S_i$.

minimize $\sum_{i=0}^{1} \sum_{s_i \in S_i} g_{s_i} - (1 - b_{s_i})$ **subject to**

Constraints 1, 2, 3, 4, 6, the domains from Formulation 1, and

$$(\forall i)(\forall s_i \in S_i) \quad g_{s_i} \geq p_{s_i} \quad (9)$$

$$(\forall i)(\forall s_i \in S_i) \quad g_{s_i} \geq 1 - b_{s_i} \quad (10)$$

When b_{s_i} is set to 0 (which can be done only when that strategy's regret is 0, by Constraint 6), g_{s_i} must be set to 1, which then cancels out with the $-(1 - b_{s_i})$ term in the objective. However, when b_{s_i} is set to 1, g_{s_i} must be set to p_{s_i} (and the $-(1 - b_{s_i})$ term in the objective will equal 0). Thus the objective is indeed to minimize the sum of the probabilities of strategies that have positive regrets.

Formulation 4: Penalize either the regret of or the probability placed on a strategy

In our final formulation, we let the solver choose to count as the penalty for a pure strategy either the strategy's (normal-

ized) regret, or probability placed on the strategy. Exactly in equilibria, every strategy has either a regret of 0 or a probability of 0. Thus these are the only solutions of the program with a total penalty of 0. The objective is the penalty plus $|S_0| + |S_1|$.

minimize $\sum_{i=0}^{1} \sum_{s_i \in S_i} f_{s_i} + g_{s_i}$ **subject to**

Constraints 1, 2, 3, 4, the domains from Formulation 1, and

$$(\forall i)(\forall s_i \in S_i) \quad f_{s_i} \geq r_{s_i}/U_i \tag{11}$$

$$(\forall i)(\forall s_i \in S_i) \quad f_{s_i} \geq b_{s_i} \tag{12}$$

$$(\forall i)(\forall s_i \in S_i) \quad g_{s_i} \geq p_{s_i} \tag{13}$$

$$(\forall i)(\forall s_i \in S_i) \quad g_{s_i} \geq 1 - b_{s_i} \tag{14}$$

If $b_{s_i} = 0$, then f_{s_i} must equal r_{s_i}/U_i (which is at most 1) and g_{s_i} must equal 1. On the other hand, if $b_{s_i} = 1$, then f_{s_i} must equal 1 and g_{s_i} must equal p_{s_i}. Thus, $f_{s_i} + g_{s_i}$ is at least 1 for every s_i, and an additional penalty must be paid either for the normalized regret of the strategy, or the probability of the strategy. Thus $f_{s_i} + g_{s_i}$ can equal 1 if and only if the strategy has either no probability or no regret; otherwise, $f_{s_i} + g_{s_i} > 1$.

Example

To illustrate the differences between the formulations, consider the following game in which $\epsilon > 0$ is small.

	L	R
U	$1, \epsilon$	$1, 0$
D	$0, 0$	$0, 1$

U strictly dominates D, and L is a strictly better response to U than R. Thus (U, L) is the unique equilibrium. Hence it is the unique feasible solution for Formulation 1, and the only optimal solution for Formulations 2–4.

Now consider the pair of strategies (U, R). The regret for playing R is only ϵ. It follows that this is a near-optimal solution for Formulation 2. For Formulation 3, however, this is not a good solution because all of the column player's probability is on a strategy with positive regret. For Formulation 4, we can choose to count the regret for playing R as the penalty, and hence (U, R) is near-optimal.

Finally, consider the pair of mixed strategies where the row player plays U with probability $1 - \epsilon$ and D with probability ϵ, and the column player plays R. The regret for playing R is now 0 (it yields an expected utility of ϵ, whereas L yields an expected utility of $(1 - \epsilon)\epsilon < \epsilon$). However, the regret for playing D is 1, and therefore this is not a good solution for Formulation 2. For Formulation 3 this is near-optimal because only ϵ probability is placed on a strategy with regret. For Formulation 4, we can choose to count the regret for playing D as the penalty; thus it is near-optimal.

Variations on the formulations

Numerous variations on the above formulations are possible. For example, in the formulations with an objective, it is possible to place different weights on the strategies in the objective. Moreover, it is in fact possible to mix up the formulations to obtain a new formulation, using one of the original formulations for some pure strategies and another one for others. We leave investigating the performance of weighted and mixed formulations for future research.

Design dimensions of *MIP Nash*

There are several design dimensions to MIP search algorithms, and in this section we study how a MIP-based equilibrium-finding algorithm should be designed along those dimensions. We implemented the variants in CPLEX 9.0, a commercial MIP software package (ILOG Inc 2003). The solving method in CPLEX is a branch-and-bound algorithm with several sophisticated techniques incorporated, which we evaluate below. We tested the algorithms on the leading test suite of game generators, GAMUT (Nudelman *et al.* 2004). That library of 24 game generators was constructed from the descriptions of many different kinds of games in the literature. Also, GAMUT is the data that was used in the prior *PNS* experiments (Porter, Nudelman, & Shoham 2004). All the experiments referred to in this section, and the next, concern the problem of finding one (any) Nash equilibrium. Therefore in those experiments we stop the search algorithm when the first equilibrium is found.

Objective function to help bias the search

Although Formulation 1 need not have an objective function, we found that adding an objective function—to guide the search—drastically speeds up the algorithm. We tried several objectives: minimizing support size, maximizing support size, minimizing welfare, maximizing welfare, minimizing the difference in the players' support sizes, and minimizing a hybrid objective consisting of the size of the supports and the difference in the players' support sizes. Our experiments showed that using *any* of the objective functions led to an order of magnitude speed improvement (over not using an objective function). The objectives of minimizing support sizes and maximizing welfare led to the best performance. Therefore, in the experiments in the rest of this paper (except where noted), we use the welfare-maximization objective.

Search (node selection) strategy

For any branch-and-bound algorithm there is a choice of which node to expand next. CPLEX provides several options including *depth-first search* (in which the algorithm chooses the "most recently created node"), *best-bound search* (in which the algorithm chooses the "node with the best objective function for the associated linear program (LP) relaxation"),[2] and *best-estimate search* (in which the algorithm chooses the node with the "best estimate of the integer objective value that would be obtained from a node once all integer infeasibilities are removed") (ILOG Inc 2003). The latter is designed specifically for problems where finding a feasible solution is difficult. As thus suspected, we observed that it had the best performance on the problem of finding an equilibrium, and we thus use that strategy for that problem. (On the problem of finding an *optimal* equilibrium, discussed later, we use best-bound search because it is designed for finding a provably optimal solution using the smallest possible search tree.)

[2]This is like A* except that a node's h-value is only approximately computed when inserting the node onto the open list; the exact computation is postponed until popping the node off the list.

Primal heuristics at nodes

At each node in the search tree, *primal heuristics* can be used to try to obtain a feasible solution or a better feasible solution. Its value will allow more pruning in that subtree. CPLEX does this by attempting to generate an integer feasible solution using information about the node's LP relaxation. We discovered that completely disabling this heuristic resulted in a 6% average speed improvement. We therefore disabled it for the rest of the experiments.

Problem formulation

Table 1 shows that Formulation 1 performed significantly better than the other three formulations (Formulation 3 was second best). Although the best formulation depends on the distribution, we use Formulation 1 for the rest of our experiments in order to conduct a fair comparison (where the algorithm is not changed across distributions) against other algorithms.

	Form1	Form2	Form3	Form4
BertrandOligopoly	286.54	196.55	**0.00**	0.16
BidirectionalLEG_CG	22.52	140.41	**14.45**	99.83
BidirectionalLEG_RG	2.35	58.14	2.18	**1.24**
BidirectionalLEG_SG	0.13	1.52	**0.12**	0.47
CovariantGame_Pos	**0.47**	464.54	28.67	464.54
CovariantGame_Rand	**203.87**	464.54	257.82	464.54
CovariantGame_Zero	**99.91**	464.54	263.16	464.54
DispersionGame	0.01	0.01	0.01	0.01
GraphicalGame_RG	**127.96**	293.82	146.81	286.99
GraphicalGame_Road	**151.18**	464.54	288.29	464.54
GraphicalGame_SG	**181.98**	464.54	246.53	464.54
GraphicalGame_SW	**234.56**	464.54	253.34	464.54
LocationGame	0.45	0.83	**0.38**	1.52
MinimumEffortGame	0.04	**0.01**	0.83	0.02
PolymatrixGame_CG	**76.80**	148.44	80.48	146.58
PolymatrixGame_RG	**42.70**	101.18	44.02	99.51
PolymatrixGame_Road	7.03	58.59	**2.05**	51.37
PolymatrixGame_SW	85.83	146.88	**68.37**	146.41
RandomGame	**168.32**	464.54	253.08	464.54
TravelersDilemma	0.05	**0.01**	**0.01**	0.03
UniformLEG_CG	0.81	0.74	**0.14**	0.19
UniformLEG_RG	2.60	41.44	1.37	**0.83**
UniformLEG_SG	**0.16**	3.83	2.60	1.67
WarOfAttrition	**0.03**	4.51	1.78	0.19
OVERALL:	**1696.29**	4448.71	1956.51	4088.82

Table 1: *Average time (in seconds) to find an equilibrium using the different MIP formulations, in* 150×150 *games from the GAMUT distributions (10 instances of each). If an instance reached the 600 second limit, that time was counted toward the average.*

Branching strategy

We also developed several strategies for choosing the next variable to branch on that are motivated by game-theoretic considerations (beyond CPLEX's default strategy which is close to the standard approach of branching on a variable with the most fractional LP value):

1. Look at the number of strategies currently selected for each player. Branch on a most fractional strategy for the player with the smallest support. If the supports are of equal size, let CPLEX choose.
2. Suppose player 1 is playing the strategy that corresponds to the LP relaxation at this node. Find a pure strategy for player 0 that is a best response to that. Swapping roles, do the same to find a pure strategy for player 1. Branch on the strategy of the two that has greatest utility improvement for the corresponding agent compared to that agent's mixed strategy at that node.
3. Same as 2, except choose the strategy from the two that gives the opponent the biggest gain.
4. Suppose the opponent plays randomly among his strategies that have not been branched out. Choose a best response (pure) strategy. Do this for both players. Branch on the strategy of the two that has greatest utility improvement for the agent compared to the agent's mixed strategy at that node.
5. Same as 4, except choose the strategy that gives the opponent the biggest gain.
6. Make $n + 1$ CPLEX calls. Call $i \in \{1, \ldots n\}$ has the constraint that the support size for each player equals i (i.e., $|support_0| = |support_1| = i$). The last call has $|support_0| < |support_1|$ vs. $|support_0| > |support_1|$ as the first branch, and the MIP objective is to minimize the sum of the support sizes.
7. Same as 6, except the objective is to maximize welfare now also in the last call. (Of course, we still stop with the first solution found.)
8. Make two CPLEX calls. In the first, $|support_0| = |support_1|$. The second call has $|support_0| < |support_1|$ vs. $|support_0| > |support_1|$ as the first branch, and the MIP objective is to minimize the sum of the support sizes.
9. Like 8, but the objective in both calls maximizes welfare.

Strategies 1 and 6-9 direct the search towards finding equilibrium with balanced supports. Strategies 2-5 are motivated by the fact that strategies are mutual best responses in equilibrium. Strategies 6 and 7 are geared towards finding equilibrium with small supports.

Table 2 Left shows the performance of these branching strategies. While each of them helped significantly on some of the distributions, each of them was slower than CPLEX's default when averaged over all GAMUT distributions (strategy 1 did not hurt that much). Therefore, in the rest of the experiments we use CPLEX's default branching strategy.

Cutting planes

Branch-and-cut is a modern, widely applied algorithm for solving MIPs (Padberg & Rinaldi 1987). It is like branch-and-bound, except that in addition, the algorithm may generate *cutting planes* (Nemhauser & Wolsey 1999). These are constraints that, when added to the problem at a search node, result in a tighter LP polytope (while not cutting off the optimal integer solution) and thus a tighter LP bound. The tighter bound in turn can cause earlier termination of the search path, yielding smaller search trees. On the other hand, more effort is invested per node to generate cuts and solve the larger LP.

It is well known that on a given problem type, different cuts can help or hurt speed. CPLEX supports nine cut families (ILOG Inc 2003): clique cuts, cover cuts, disjunctive cuts, flow cover cuts, flow path cuts, generalized upper bounding cover cuts, implied bound cuts, Gomory fractional cuts, and mixed integer rounding cuts. We experimented with them by enabling only one at a time. We compared the performance to default CPLEX, which has them all on. CPLEX always determines heuristically which cuts to use from the families that are enabled.

	Default	1	2	3	4	5	6	7	8	9	Lemke-Howson	PNS
BertrandOligopoly	286.54	242.11	249.28	69.99	65.57	24.97	**1.50**	10.12	40.90	284.50	0.04	**0.01**
BidirectionalLEG-CG	22.52	36.69	91.10	53.34	103.73	103.81	**12.22**	19.32	75.05	27.40	0.06	**0.01**
BidirectionalLEG-RG	2.35	2.08	15.04	4.55	62.16	3.92	**0.86**	5.36	25.06	1.89	0.05	**0.01**
BidirectionalLEG-SG	0.13	51.03	3.81	5.87	51.68	51.92	0.24	0.63	1.02	**0.02**	0.06	**0.01**
CovariantGame-Pos	**0.47**	1.04	1.60	1.33	1.98	1.89	5.36	0.57	75.45	1.44	0.06	**0.01**
CovariantGame-Rand	203.87	233.63	240.50	242.79	245.62	248.50	213.01	**195.32**	326.24	239.66	376.92	**267.81**
CovariantGame-Zero	**99.91**	135.80	251.57	252.19	360.94	290.64	365.00	149.31	449.82	355.01	263.48	**0.13**
DispersionGame	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.05	**0.01**
GraphicalGame-RG	**127.96**	215.47	269.70	298.82	282.38	229.96	247.29	216.85	277.82	287.30	96.02	**0.05**
GraphicalGame-Road	**151.18**	195.18	272.72	244.48	287.83	271.30	381.20	266.26	378.58	454.29	277.80	**0.13**
GraphicalGame-SG	**181.98**	247.61	313.96	311.61	365.76	343.50	464.54	322.48	464.54	458.68	133.07	**0.10**
GraphicalGame-SW	**234.56**	331.60	342.45	316.74	446.92	420.09	413.59	275.85	453.39	464.54	168.49	**0.09**
LocationGame	0.45	0.45	**0.01**	**0.01**	**0.01**	**0.01**	0.05	0.84	0.27	3.81	0.05	**0.01**
MinimumEffortGame	0.04	0.04	**0.02**	**0.02**	**0.02**	**0.02**	0.55	0.05	0.36	0.26	0.05	**0.01**
PolymatrixGame-CG	**76.80**	107.17	95.61	85.03	146.25	101.95	139.16	100.14	147.04	100.77	72.82	**65.13**
PolymatrixGame-RG	42.70	**28.63**	99.09	87.02	99.09	61.50	56.00	68.13	99.11	60.23	76.26	**0.01**
PolymatrixGame-Road	**7.03**	50.95	50.95	50.95	50.95	50.95	51.29	51.12	71.83	53.65	1.26	**0.05**
PolymatrixGame-SW	85.83	79.76	114.19	91.72	103.37	119.23	146.34	**42.06**	70.03	109.83	145.38	**0.13**
RandomGame	**168.32**	304.03	322.18	333.13	343.77	364.30	366.90	291.92	464.54	464.54	162.08	**0.16**
TravelersDilemma	0.05	0.06	0.05	**0.04**	**0.04**	**0.04**	3.63	18.35	36.94	17.60	0.02	**0.01**
UniformLEG-CG	0.81	0.39	0.39	0.49	0.50	0.27	**0.16**	1.04	0.83	0.84	0.05	**0.01**
UniformLEG-RG	2.60	1.37	10.80	2.60	8.88	2.82	**1.24**	3.62	13.02	3.52	0.05	**0.01**
UniformLEG-SG	**0.16**	1.04	7.70	2.80	7.31	3.11	1.04	2.29	5.12	0.81	0.05	**0.01**
WarOfAttrition	0.03	0.15	1.37	**0.02**	0.50	**0.02**	0.04	5.83	0.04	0.05	4.29	**0.01**
OVERALL:	**1696.29**	2266.29	2754.13	2455.56	3035.26	2694.73	2871.21	2047.48	3477.03	3390.67	1778.50	**333.94**

Table 2: *Average time to find an equilibrium in* 150×150 *games (10 instances). Left: branching strategies. Right: Lemke-Howson and PNS. The percentage of time-outs (10 minute limit) for MIP Nash, Lemke-Howson, and PNS was 7.5%, 8.3%, and 2.0%, respectively.*

There was significant variability as to which cuts hurt or helped on different GAMUT distributions (we omit the complete results table due to lack of space). Interestingly, using no cuts, or using any one cut family alone, was 16% faster on average than CPLEX's defaults. The exceptions were the last two families, which were within 3% of the default speed.

Experiments on finding a Nash equilibrium

In this section we compare the performance of *MIP Nash* against the prior state-of-the-art algorithms: *Lemke-Howson* and *PNS*. We used the implementation of *Lemke-Howson* available in the Gambit software library (McKelvey, McLennan, & Turocy 2004). For *PNS*, we used code given to us by its authors.[3] Table 2 shows the performance on the GAMUT distributions. *MIP Nash* was faster than *Lemke-Howson*, but not nearly as fast as *PNS*.

Most of the games generated by the GAMUT distributions have equilibria with small (and balanced) supports. This is supported theoretically as it is known that most n-player games with payoffs generated uniformly at random from the n-dimensional unit sphere have at least one equilibrium with small supports (McLennan & Berg 2005). The speed of *PNS* on these distributions is largely due to its bias of searching through (balanced) supports in smallest-first order. However, there is no guarantee that real-world games are generated by such distributions. For example, (an n-strategy generalization of) the rock-paper-scissors game only has an equilibrium where all pure strategies are played with equal probability. On that type of game, *PNS* would have to search through all smaller supports before finding an equilibrium, thus making it prohibitively slow. Of course, one could use an algorithm with the reverse bias (searching through large supports first), or even an algorithm that interleaves searches with these two biases, thus performing well on games that have equilibria with small or large supports. Still the algorithm would do poorly on games with medium-sized supports. In fact, there are too many medium-sized supports to exhaustively search through (even just considering supports of size $|S_i|/2$, there are $\binom{|S_i|}{|S_i|/2} \geq 2^{|S_i|/2}$ of them for each agent i). Therefore, we argue that in order to evaluate how well an algorithm can really capitalize on the structure of any equilibrium-finding problem (rather than testing whether the distribution is amenable to a particular rigid search bias), experiments should also be conducted on games that only have equilibria of medium-sized supports.

Furthermore, those test games should not allow for many of the game's strategies to be eliminated using dominance or iterated dominance. Moreover, this should remain the case even after branching some strategies out of the support.

To conduct such experiments, we introduce a family of games that satisfy both of these properties. For any positive integer k, the game G_k has actions $a_1, \ldots, a_{2k-1}, b_1, \ldots b_{2k}$ for the row player and actions $c_1, \ldots, c_{2k-1}, d_1, \ldots d_{2k}$ for the column player. The utilities are:

- $u(a_i, c_{i+1 (\mathrm{mod}\ 2k-1)}) = (2, 4)$, $u(a_i, c_{i-1 (\mathrm{mod}\ 2k-1)}) = (4, 2)$
- $u(a_i, c_j) = (3, 3)$ for $j \notin \{i \pm 1 (\mathrm{mod}\ 2k-1)\}$
- $u(a_i, d_j) = (2, 0)$, $u(b_i, c_j) = (0, 2)$, $u(b_i, d_i) = (3, 0)$
- $u(b_i, d_{i+1}) = (0, 3)$ for odd i, $u(b_i, d_{i-1}) = (0, 3)$ for even i
- $u(b_i, d_j) = (0, 0)$ otherwise.

$Example: G_2$	c_1	c_2	c_3	d_1	d_2	d_3	d_4
a_1	3, 3	2, 4	4, 2	2, 0	2, 0	2, 0	2, 0
a_2	4, 2	3, 3	2, 4	2, 0	2, 0	2, 0	2, 0
a_3	2, 4	4, 2	3, 3	2, 0	2, 0	2, 0	2, 0
b_1	0, 2	0, 2	0, 2	3, 0	0, 3	0, 0	0, 0
b_2	0, 2	0, 2	0, 2	0, 3	3, 0	0, 0	0, 0
b_3	0, 2	0, 2	0, 2	0, 0	0, 0	3, 0	0, 3
b_4	0, 2	0, 2	0, 2	0, 0	0, 0	0, 3	3, 0

Proposition 1 *G_k has a unique equilibrium. Every a_i and c_i is played w.p. $1/(2k-1)$. The b_i and d_i are never played.*

Proof: First, suppose that some strategy b_i (with i odd) were sometimes played in equilibrium. In order for b_i to perform at least as well as a_1, it must be the case that d_i is played

[3] We are aware of at least one instance of a 2×2 game (matching pennies) in which this code returns an incorrect result. We do not know of any larger games where the code is incorrect.

with probability at least $2/3$. In order for d_i to perform at least as well as c_1, it must be the case that b_{i+1} is played with probability at least $2/3$. In order for b_{i+1} to perform at least as well as a_1, it must be the case that d_{i+1} is played with probability at least $2/3$. But, it is impossible for each of d_i and d_{i+1} to be played with probability $2/3$. It follows that no strategy b_i (with i odd) is ever played in equilibrium, and similarly it can be shown that no strategy b_i (with i even), d_i (with i odd), or d_i (with i even) is ever played in equilibrium.

The remainder of the game (the strategies a_i and c_i) constitute a symmetric zero-sum game. Thus, each player is able to guarantee herself an expected payoff of 3. Now, if the row player plays strategy a_i with positive probability p, she should play strategy $a_{i+2(\mathrm{mod}\ 2k-1)}$ with probability at least p (otherwise the column player could get expected utility greater than 3 by playing $c_{i+1(\mathrm{mod}\ 2k-1)}$). By repeated application of this (and the fact that $2k-1$ is odd), it follows that each a_i must be played with the same probability in equilibrium. Symmetrically, the same holds for the c_i. ■

Figure 1 shows that on this game, *MIP Nash* drastically outperforms *PNS*. The reason is that *MIP Nash* has a flexible search bias and the techniques of MIP guiding it (such as preprocessing, LP-guided branch selection, LP-based bounding, cutting planes, and objective-based guidance). *MIP Nash* is not as fast as *Lemke-Howson* on this game.

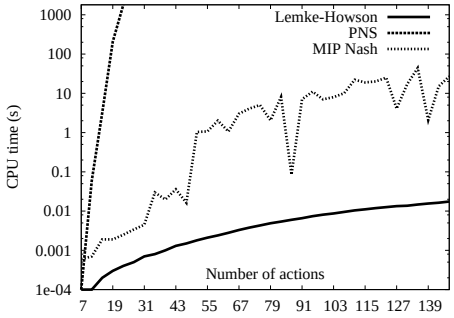

Figure 1: *Finding the Nash equilibrium in the game G_k.*

Anytime algorithms for ϵ-equilibrium

A pair of mixed strategies is an *ϵ-equilibrium* if the regret of each player's mixed strategy is at most ϵ. In this subsection, we show that feasible solutions to Formulations 2, 3, and 4 with low objective values constitute ϵ-equilibria. Thus, applying a MIP solver to any one of these three formulations constitutes an *anytime algorithm* for finding ϵ-equilibrium, which will eventually return an equilibrium ($\epsilon = 0$).[4]

Proposition 2 *In a feasible solution to Formulation 2 with objective value ϵ, the sum of the players' regrets is at most ϵ.*

Proof: Let $\delta(0) = 0$, $\delta(x) = 1$ for $x > 0$. The sum of the regrets is $\sum_{i=0}^{1} \sum_{s_i \in S_i} p_{s_i} r_{s_i} \leq \sum_{i=0}^{1} \sum_{s_i \in S_i} \delta(p_{s_i}) r_{s_i}$, which equals the objective value of Formulation 2. ■

[4] ϵ-equilibria have also been studied before. There always exists an ϵ-equilibrium where both players randomize over at most $\frac{12 \ln n}{\epsilon^2}$ strategies (where n is the number of an agent's pure strategies). Thus one can find an ϵ-equilibrium (by searching over all such small supports) in $n^{O(\ln n)}$ time—for a *given fixed* ϵ (Lipton, Markakis, & Mehta 2003).

Proposition 3 *In a feasible solution to Formulation 3 with objective value $\epsilon/\overline{U}$, the sum of the players' regrets is at most ϵ, where $\overline{U} = \max\{U_0, U_1\}$.*

Proof: The sum of the regrets is $\sum_{i=0}^{1} \sum_{s_i \in S_i} p_{s_i} r_{s_i} \leq \sum_{i=0}^{1} \sum_{s_i \in S_i} p_{s_i} \delta(r_{s_i}) \overline{U}$, which is $\overline{U}$ times the Formulation 3 objective. ■

Proposition 4 *In a feasible solution to Formulation 4 with objective value $(\epsilon/\overline{U}) + |S_0| + |S_1|$, the sum of the players' regrets is at most ϵ, where $\overline{U} = \max\{U_0, U_1\}$.*

Proof: The sum of the regrets is $\sum_{i=0}^{1} \sum_{s_i \in S_i} p_{s_i} r_{s_i} \leq \sum_{i=0}^{1} \sum_{s_i \in S_i} \min\{p_{s_i}, r_{s_i}/\overline{U}\}\overline{U}$, which is at most $\overline{U}$ times the value obtained by subtracting $|S_0| + |S_1|$ from the Formulation 4 objective. ■

Finding an *optimal* Nash equilibrium

Finding an equilibrium may not be satisfactory: we may want to optimize some *objective* over the space of equilibria. Perhaps the most natural objective is *social welfare*, i.e., the sum of the agents' utilities. (Note that any social welfare maximizing equilibrium is also Pareto optimal within the space of Nash equilibria.) Other objectives are also possible: we may wish to maximize one of the players' utilities, maximize the minimum utility between the agents, minimize the support sizes of the equilibrium strategies, etc. (Each of those problems is $\mathcal{NP}$-complete (Gilboa & Zemel 1989; Conitzer & Sandholm 2003).) In this section we show how *MIP Nash* can be used to find an optimal Nash equilibrium.

Optimizing using Formulation 1

Formulation 1 is especially well-suited to optimizing objectives because it does not have an objective of its own. Thus, for social welfare, we can simply add the objective **maximize** $u_0 + u_1$. Other linear objectives are similarly easy to optimize.

Some nonlinear objectives, such as the minimum of two expressions or the absolute value of an expression, can be used. For example, the minimum utility between the agents can be maximized by adding the constraints $r \leq u_0$ and $r \leq u_1$, and maximizing r. As another example, the difference between the players' utilities can be minimized (to minimize envy) by adding the constraints $r \geq u_0 - u_1$ and $r \geq u_1 - u_0$, and minimizing r.

The objective does not have to be concerned with utilities. It can also involve support sizes, probabilities of strategies being played, etc., and nonlinear objectives involving these variables (analogous to those just discussed involving utilities). Formulation 1 can be used to optimize these as well.

Optimizing using Formulations 2, 3, and 4

Formulations 2, 3, and 4 are not as well suited to optimizing an objective because they already use the MIP objective in the specification of equilibrium. One solution is to add the desired objective to the existing objective with a small coefficient. For example, we may change the objective of Formulation 2 to

$$\textbf{minimize} \left(\sum_{i=0}^{1} \sum_{s_i \in S_i} f_{s_i} - U_i b_{s_i} \right) - w(u_0 + u_1),$$

for some constant w, in an attempt to maximize social welfare over the space of equilibria. However, if w is not chosen small enough, the solver may choose to sacrifice the equilibrium property (shifting to an ϵ-equilibrium instead) to obtain higher social welfare. One technique for dealing with this is to repeatedly decrease (say, halve) the value of w until an equilibrium is produced by the solver. Once an equilibrium is obtained by this method, it must indeed optimize the desired objective (because all equilibria have the same value for the formulation's original objective). However:

Proposition 5 *To optimize social welfare in Formulations 2, 3, or 4 using the technique described above, arbitrarily small settings of w can be required (even in 2×2 games).*

Experiment on finding an optimal equilibrium

This section studies how *MIP Nash* (Formulation 1) performs compared to the prior algorithms on finding an optimal equilibrium. Neither *Lemke-Howson* nor *PNS* were designed to optimize an objective. There are, however, methods of using these algorithms to find all equilibria, from which the optimal one can be selected. We configured *PNS* to search all supports (which results in an algorithm similar to Dickhaut-Kaplan (Dickhaut & Kaplan 1991)).[5] To evaluate *Lemke-Howson*, we use a variant, by Mangasarian, that enumerates all equilibria (Mangasarian 1964) (we refer to it as M-Enum). Table 3 shows that *MIP Nash* outperforms the other algorithms by 2-3 orders of magnitude.

actions	*M-Enum*	*PNS*	*MIP Nash*
10	2.21 (0%)	26.45 (3.7%)	0.001 (0%)
25	429.14 (66.7%)	600 (100%)	3.01 (0%)
50	425.07 (66.7%)	600 (100%)	30.44 (4.2%)

Table 3: *Average time (in seconds), over all GAMUT distributions (6 instances of each), for finding a welfare-maximizing equilibrium. The percentage of timeouts (limit here was 600s) is in parentheses.*

Conclusions and future research

We presented MIP formulations for finding Nash equilibria in two-player games. We studied different design dimensions of search algorithms that are based on those formulations. On the problem of finding one (any) equilibrium, *MIP Nash* outperforms *Lemke-Howson* but not *PNS* on GAMUT data. We argued that experiments should also be conducted on games with equilibria with medium-sized supports only, and presented a methodology for generating such games. On such games *MIP Nash* drastically outperforms *PNS* but not *Lemke-Howson*. *MIP Nash* Formulations 2, 3, and 4 also yield anytime algorithms for ϵ-equilibrium, with provable bounds. Another advantage of *MIP Nash* is that it can be used to find an *optimal* equilibrium (according to various objectives). The prior algorithms can be extended to that setting, but they are orders of magnitude slower.

Future research includes developing MIP-based search algorithms for restricted games and for structured representations of games. Future research also includes extending the MIP approach to games with more than two players. This is not straightforward even for Nash equilibrium, and furthermore, solution concepts that take coalitional deviations into account (Aumann 1959; Bernheim, Peleg, & Whinston 1987) may be more appropriate in that context.

[5] Actually, when a pair of supports has multiple equilibria associated with it, PNS will only find one. Correcting for this would increase the search time even further.

References

Antipin, A. 2003. Extragradient approach to the solution of two person non-zero sum games. *Optimization and Optimal Control*, World Scientific. 1–28.

Aumann, R. 1959. Acceptable points in general cooperative n-person games. volume IV of *Contributions to the Theory of Games*. Princeton University Press.

Bernheim, B. D.; Peleg, B.; and Whinston, M. D. 1987. Coalition-proof Nash equilibria: I concepts. *Journal of Economic Theory* 42(1):1–12.

Conitzer, V., and Sandholm, T. 2003. Complexity results about Nash equilibria. In *IJCAI-03*, 765–771.

Dickhaut, J., and Kaplan, T. 1991. A program for finding Nash equilibria. *The Mathematica Journal* 87–93.

Gilboa, I., and Zemel, E. 1989. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior* 1(1):80-93.

ILOG Inc. 2003. CPLEX 9.0 User's Manual.

Khamisov, O. 2003. A global optimization approach to solving equilibrium programming problems. *Optimization and Optimal Control*. World Scientific. 155–164.

Lemke, C., and Howson, J. 1964. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics* 12:413–423.

Lipton, R.; Markakis, E.; and Mehta, A. 2003. Playing large games using simple strategies. In *ACM-EC*, 36–41.

Mangasarian, O. 1964. Equilibrium points in bimatrix games. *Journal of the Society for Industrial and Applied Mathematics* 12(4):778–780.

McKelvey, R. D.; McLennan, A. M.; and Turocy, T. L. 2004. Gambit: Software tools for game theory, version 0.97.1.5.

McLennan, A., and Berg, J. 2005. The asymptotic expected number of Nash equilibria of two player normal form games. *Games and Economic Behavior*. Forthcoming.

Myerson, R. 1991. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge.

Nash, J. 1950. Equilibrium points in n-person games. *Proc. of the National Academy of Sciences* 36:48–49.

Nemhauser, G., and Wolsey, L. 1999. *Integer and Combinatorial Optimization*. John Wiley & Sons.

Nudelman, E.; Wortman, J.; Leyton-Brown, K.; and Shoham, Y. 2004. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS-04*.

Padberg, M., and Rinaldi, G. 1987. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters* 6:1–7.

Papadimitriou, C. 2001. Algorithms, games and the Internet. In *STOC*, 749–753.

Porter, R.; Nudelman, E.; and Shoham, Y. 2004. Simple search methods for finding a Nash equilibrium. In *AAAI-04*, 664–669.

Savani, R., and von Stengel, B. 2004. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *FOCS*.