

Algorithms for Rationalizability and CURB Sets*

Michael Benisch, George Davis and Tuomas Sandholm

School of Computer Science

Carnegie Mellon University

5000 Forbes Ave.

Pittsburgh, PA 15213

{mbenisch, gbd, sandholm}@cs.cmu.edu

Keywords: algorithms, rationalizability, CURB sets, Nash equilibrium, computer science

JEL classifications: C63 - Computational Techniques, C72 - Noncooperative Games

Abstract

Significant work has been done on computational aspects of solving games under various solution concepts, such as Nash equilibrium, subgame perfect Nash equilibrium, correlated equilibrium, and (iterated) dominance. However, the fundamental concepts of rationalizability and CURB (Closed Under Rational Behavior) sets have not, to our knowledge, been studied from a computational perspective. First, for rationalizability we describe an LP-based polynomial algorithm that finds all strategies that are rationalizable against a mixture over a given set of opponent strategies. Then, we describe a series of increasingly sophisticated polynomial algorithms for finding all minimal CURB sets, one minimal CURB set, and the smallest minimal CURB set. Finally, we give theoretical results regarding the relationships between CURB sets and Nash equilibria, showing that finding a Nash equilibrium can be exponential only in the size of the smallest CURB set. We show that this can lead to an arbitrarily large reduction in the complexity of finding a Nash equilibrium. On the downside, we also show that the smallest CURB set can be arbitrarily larger than the supports of the enclosed Nash equilibrium.

*This material is based upon work supported by National Science Foundation ITR grant 0205435, IGERT grant 9972762, and IIS grants 0121678 and 0427858, as well as Office of Naval Research grant N00014-02-1-0973, and a Sloan Fellowship. Additional support at Carnegie Mellon University was provided by the Agent-Mediated Electronic Marketplaces Laboratory, the Center for Computational Analysis of Social and Organizational Systems, and the e-Supply Chain Management Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the National Science Foundation, the Office of Naval Research, or the U.S. government. A short early version of this paper appeared in the National Conference on Artificial Intelligence (AAAI), 2006. We thank Vincent Conitzer and Andrew Gilpin for their helpful input and guidance.

1 Introduction

For multi-agent systems, game-theoretic solution concepts help agents choose strategies, help modelers predict system states, and help mechanism designers guarantee properties of the games they create. Significant attention has been given to algorithms for finding equilibria according to the solution concepts of subgame perfect Nash equilibrium (e.g., minimax search and α - β -pruning), Nash equilibrium [13, 17, 19], correlated equilibrium [10], and more recently, iterative dominance [12, 6] and related concepts [7].

Nash equilibrium (under which no agent has incentive to deviate from its mixed strategy given that the other agents do not deviate from theirs) remains the most important point-valued solution concept. However, it has been recently shown that finding a Nash equilibrium, even in two player normal-form games with payoffs in $\{0, 1\}$, is PPAD-complete [5, 1], suggesting that no polynomial-time algorithms exist for the problem.

There are other fundamental solution concepts that have certain known advantages over Nash equilibrium, and—as we will show—solutions according to those concepts can be found in polynomial time even in the worst case. Specifically, we will study the concept of *rationalizability* and the concept of the *minimal Closed Under Rational Behavior (CURB) set* [2] for two-player normal-form games. A game can have multiple Nash equilibria, but each of those is a point, i.e., a strategy profile. In contrast, in these concepts a solution is a *set* of strategy profiles.

The notion of rationalizability was introduced by Pearce [16] and Bernheim [4]. It has by now been extensively used as a robust solution concept in game theory, and in applications such as auctions (e.g., [3]). Its main insight is that rationality restricts players from ever playing strategies that are not best responses given the beliefs they hold about their opponents; strategies that are not best responses to a set of consistent beliefs about opposing strategies are not *rationalizable*. The process of iteratively removing strategies that are dominated (potentially by mixtures) captures this by emulating player assumptions that opponents will never play strategies which are not best responses to one of their strategies [16].

The set of all players’ rationalizable strategies has the property that no player’s best response to a mixture over strategies inside the set lies outside the set – in other words, the set is Closed Under Rational Behavior (or CURB). However, this set may have subsets which are also CURB, illustrating that players with mutually consistent prior beliefs about which subset of strategies their opponents might play can rationally discard more strategies than is possible via iterative elimination, even in two-player games.

Formally, a CURB set, S , is defined as a set of strategies that contains the best responses to any mixture over itself: if S is CURB and players believe that no strategy outside of S will be played with positive probability by their opponents, then such strategies will indeed not be played by rational players. Formally a CURB set is defined by the following operators.

- $\beta_i(m)$: a function that returns player i ’s best responses to the mixed strategy m .
- $M(S)$: a function that returns all possible mixtures with supports in S .
- $\beta_i(S)$: a function that returns player i ’s best responses to any mixture with supports

in S ,

$$\beta_i(S) = \bigcup_{m \in M(S)} \beta_i(m)$$

- $\beta(S)$: the Cartesian product of the sets $\beta_i(S)$ over all players, i .

$$\beta(S) = \bigcup_i \beta_i(S)$$

Under this notation, a set, S , is CURB if $\beta(S) \subset S$. The entire game is trivially CURB by this definition. Basu and Weibull distinguish a *minimal* CURB set as a CURB set that does not contain any CURB subsets [2].

The minimal CURB set solution concept has been motivated from several perspectives, including the following:

- As is well known, nonstrict mixed Nash equilibria can be highly unstable because a player is indifferent between (some of) his pure strategies. Strict Nash equilibria would be stable but many games lack such equilibria. Minimal CURB sets are the “nearest set-valued generalization of strict Nash equilibria” (it is the smallest set of strategies that includes all ways of choosing among the indifferences) and a solution according to this concept is guaranteed to exist [2].
- Any CURB set can be viewed as a subspace of strategies within which any best-response dynamic (even a best-response dynamic of mixed strategies) will stay within. CURB sets have thus been examined as a solution concept that describes the strategy subspace where iteratively adapting agents will eventually settle (e.g., [11]).
- More recently, Voorneveld *et.al.* have enumerated properties of minimal CURB sets that illustrate the advantages of set-based solution concepts over point-valued concepts such as Nash equilibria [21].

In order for a solution concept to be operational, it needs to be accompanied by algorithms for solving games according to the concept. Finding minimal CURB sets has been considered a complex matter (e.g., [18]), and little work has been done on the computational complexity or algorithms for the problem. To our knowledge, the only exception is the work of Pruzhansky, which studied sequential games of perfect information. Such games are relatively simple in that they contain exactly one minimal CURB set, and a straightforward algorithm can quickly find it by exploiting the sequential representation [18]. We present the first computational treatment of CURB sets in normal form games. We show that minimal CURB sets are actually easy to find: the complexity is polynomial even in the worst case.

The rest of the paper is organized as follows. We present and analyze a family of algorithms which compute, for a two player normal form game, in time polynomial in the total number of pure strategies, the set of all rationalizable strategies, all minimal CURB sets, a single minimal CURB set, and the smallest minimal CURB set. Finally, we discuss additional applications of our results, including the potential of finding minimal CURB sets to bound the computation involved in finding Nash equilibria.

2 Finding Rationalizable Strategies

Finding strategies that are rationalizable against a mixture over a given set of opponent strategies is a problem of interest in its own right, and plays a central role in our computation of CURB sets. This section describes a polynomial complexity function, `all_rationalizable`, for doing that. The algorithm below is for the row player. The column player’s algorithm is symmetric.

function `all_rationalizable`(S_r, S_c, u_r)

$S_r^* \leftarrow \emptyset$

for each row strategy, $s_r \in S_r$ **do**

if there exists a feasible solution to the following linear feasibility problem:

find p_{s_c} **such that**

$$(1) \quad \sum_{s_c} p_{s_c} = 1$$

$$(2) \quad (\forall s'_r \in S_r / \{s_r\}) \quad \sum_{s_c} p_{s_c} u_r(s_r, s_c) \geq \sum_{s_c} p_{s_c} u_r(s'_r, s_c)$$

then $S_r^* \leftarrow S_r^* \cup s_r$

return S_r^*

The parameters to the function are a set of row player strategies to consider, S_r , a set of column player strategies they may be played against, S_c , and the row-player’s utility function, u_r . For each row strategy, $s_r \in S_r$, a linear feasibility problem (LFP) (i.e., a linear program with no objective) is constructed to find a mixture, p_{s_c} , over column player strategies such that s_r is the row player’s best response. The constraints of the LFP ensure that the mixture is valid (sums to 1) and that the row player’s utility by playing s_r against p_{s_c} is greater than or equal to that of any other strategy in S_r . If and only if the LFP has a feasible solution, s_r is added to the set of rationalizable strategies.

The computational complexity of the functions described in this paper depend on the total number of strategies in the game, which we will denote by n , and the complexity of solving a linear feasibility problem where the number of variables and the number of constraints are bounded by n , which we will denote as `LFP`(n). Linear feasibility problems can be solved in low-order polynomial time even in the worst case. They are no slower to solve than linear programs (the fastest known algorithms for LFPs are faster, in the worst case, than the fastest known linear programming algorithms [22]) because linear program solving involves solving a linear feasibility program as the first phase to find a feasible solution, after which the linear program solver needs to still improve that solution to reach an optimum. (In the experiments, we solve the LFP using the simplex algorithm, which has worst-case exponential complexity but is known to outperform polynomial linear programming algorithms in practice.)

Proposition 1. *Function `all_rationalizable` returns all rationalizable strategies, and nothing else. It is $O(n) \times LFP(n)$.*

Proof. For any strategy to be rationalizable, by definition, there must be a mixture over opponent strategies for which that strategy is a best response. By inspection, the linear program in `all_rationalizable` will return true if and only if such a mixture exists. Since `all_rationalizable` runs this program on all strategies, and includes them in the return set only if the linear program is feasible, it must be correct. Since the linear program is executed once for each strategy, and the size of the linear program is bounded by n , `all_rationalizable` has complexity as shown. \square

3 Finding CURB Sets

We now turn our attention to the problem of finding CURB sets. The function below finds a minimal set of strategies that both 1) contains a given seed strategy, s_r , and 2) is CURB. (Note that the returned set is not necessarily a minimal CURB set.) It alternates between the players, calling the `all_rationalizable` function to identify strategies that have become rationalizable via the addition of opponent strategies. If an iteration passes without strategies being added, the algorithm has converged.

```

function min_containing_CURB( $s_r, \langle S_r, S_c, u \rangle$ )
   $S_r^* \leftarrow \{s_r\}, S_c^* \leftarrow \emptyset$ 
  converged  $\leftarrow$  false
  while  $\neg$ converged do
    converged  $\leftarrow$  true
    for  $(p, o) \in [(c, r), (r, c)]$  do
       $S'_p \leftarrow$  all_rationalizable( $S_p \setminus S_p^*, S_o^*, u_p$ )
      if  $S'_p \neq \emptyset$  then
        converged  $\leftarrow$  false
         $S_p^* \leftarrow S_p^* \cup S'_p$ 
  return sub-game,  $G' = \langle S_r^*, S_c^*, u \rangle$ 

```

Proposition 2. *Function `min_containing_CURB` is $O(n^2) \times LFP(n)$.*

Proof. Every two calls made to `all_rationalizable` must add a strategy to the return set, or else `min_containing_CURB` must terminate. Since at most n strategies can be added this way, the complexity of `min_containing_CURB` is $O(n^2) \times LFP(n)$. \square

Theorem 1. *Function `min_containing_CURB` is correct, that is, the returned set, S^* , is a minimal set of strategies that both 1) contains the given seed strategy, s_r , and 2) is CURB.*

Proof. We show that S^* is CURB by considering its state after MCC converges. The convergence of the algorithm implies that no strategies outside of S^* are rationalizable with respect to S^* . Therefore, $\beta(S^*) \subset S^*$, and S^* is CURB.

To prove that S^* is the minimal containing CURB set of s_r (it contains no CURB subsets that include s_r), we will use induction on the strategies added to S^* .

- **Base Case:** Initially S^* contains only s_r and $\beta_c(s_r)$. At this point, S^* is trivially the minimal containing CURB set of s_r because we cannot remove s_r and removing the column player's best response to s_r breaks the CURB property.
- **Inductive Step:** Each time a new strategy s^* is added to S^* it is necessarily a best response to some mixture, $m \in M(S^*)$, over the strategies already contained in S^* . Since strategies are never removed from S^* during the execution of MCC, m will remain a valid mixture. Therefore, no new strategy s^* can be removed from S^* without breaking the CURB property.

□

We will present three algorithms which use the above function to detect minimal CURB sets in a game. To facilitate understanding of these algorithms, we first present three results regarding CURB set structure.

Theorem 2. *If each of two intersecting strategy sets is CURB, then their intersection is also CURB.*

Proof. Consider two CURB sets S_A and S_B with the non-empty intersection S_I . For any mixture over strategies in S_I belonging to (without loss of generality) the row player, there exists a pure strategy which is column player's best response, s_c^* . Because S_A is CURB and also contains the row mixture, $s_c^* \in S_A$; likewise $s_c^* \in S_B$. Therefore s_c^* is within their intersection, S_I . □

Since the intersection of two CURB sets must be CURB and contained in both sets, we have the following.

Corollary 1. *Minimal CURB sets cannot overlap (i.e., share any rows or columns).*

Proof. Consider minimal CURB sets S_A and S_B which share non-empty overlap S_C . By the containment theorem, S_C must also be CURB. Since, by the definition of intersection, $S_C \subset S_A$, this contradicts our predicate that S_A is minimal. □

Corollary 2. *Each row strategy belongs to at most one minimal CURB set.*

Proof. This is trivially true from Corollary 1. □

We are now ready to present the three algorithms. They are laid out in the next three subsections, respectively.

3.1 Finding All Minimal CURB Sets

The broadest query one can make regarding the minimal CURB set structure of a game is to find all minimal CURB sets. For one, this is useful in the adaptive agent context, to identify regions of strategy space into which learning agents may settle (e.g., [11]). The `all_MC` function described below answers this query.

To determine all of the minimal CURB sets, the `min_containing_CURB` function can be executed with each row strategy, in turn, as a seed. For the first seed, this call is made with the parameter $\langle S_r, S_c, u \rangle$ indicating the entire game. However, the result above regarding CURB intersections shows that each of the CURB sets returned by `min_containing_CURB` must contain the minimal containing CURB sets of each of its members. Therefore, we can accelerate future calls by maintaining a map between each strategy and the smallest CURB set in which it has been discovered so far. We use the subgame restricted to that strategy set as the parameter $\langle S_r, S_c, u \rangle$ when that strategy is used as the seed. Whenever such a call results in a smaller CURB set, we eliminate the previous CURB set from consideration, as it cannot be minimal. Once each strategy has been used as a seed, `all_MC` terminates and returns the CURB sets that have not been eliminated.

From the corollary above, the fact that `all_MC` executes `min_containing_CURB` on each row strategy, and the fact that no superset CURB sets remain, we have the following.

Proposition 3. *Function `all_MC` finds all minimal CURB sets, and nothing else. It is $O(n^3) \times LFP(n)$.*

Proof. By Corollary 1, the minimal CURB set for any strategy must either equal or be contained by any other CURB set in which the strategy is found. Therefore Theorem 1 holds even when `min_containing_CURB` is called on a smallest CURB set in which a strategy has been found so far. By this and Corollary 2, the main loop of `all_MC` must discover all minimal CURB sets in the game. Since any CURB set which is not minimal must have contained one of the minimal CURB sets discovered, it is removed when the smaller CURB set is discovered (or not added if the smaller set was previously discovered). Therefore `all_MC` returns all, and only minimal CURB sets. In the worst case, `all_MC` must call `min_containing_CURB` n times, with the full game as a parameter, giving time complexity $O(n^3) \times LFP(n)$. \square

3.2 Finding One Minimal CURB Set

Rather than finding all minimal CURB sets in a game, it may be desirable to quickly find any single minimal CURB set. To complete this query, we can use the `min_containing_CURB` function with a random strategy as the seed. Since the discovered CURB set might not be minimal, we recur within it by choosing as a seed a contained strategy that has not yet been used as a seed. We repeat this until all strategies in the current set have been used as seeds, at which point we terminate and return the remaining set. This constitutes the `one_minimal_CURB` algorithm.

If the game has more than one CURB set, `one_MC` will be faster than `all_MC` because it will never leave the smallest CURB set within which it ever chooses a seed. The exact speed of `one_MC` depends on the first seed chosen. If it happens to be in a small CURB set, `one_MC` runs faster. In the worst case where the entire game is the only CURB set, `one_MC` executes all of the same steps as `all_MC`. Due to this fact, and the fact that the returned set has been confirmed to be the minimal containing CURB set for all strategies within it, we have the following.

Proposition 4. *Function `one_MC` returns a minimal CURB set, and is $O(n^3) \times LFP(n)$.*

Proof. If there are no other minimal CURB sets, then the entire game is minimally CURB and will be therefore returned, according to Proposition 3. If there are any other minimal CURB sets, one of them will be discovered when a strategy inside it is used as a seed. In the worst case (when the whole game is minimally CURB), `one_MC` must call `min_containing_CURB` n times, with the full game as a parameter, giving time complexity $O(n^3) \times LFP(n)$. \square

3.3 Finding the Smallest Minimal CURB Set

As a different type of query, one may be interested in finding a *smallest* minimal CURB set. This is important, for example, if the CURB set is used for future computations (e.g., for Nash equilibrium finding as we will discuss later in the paper) and the complexity of those future computations increases with the size of the CURB set.

We find the smallest minimal CURB set using a pseudo-parallelization of `all_MC`, wherein we expand a candidate set only when it is one of the smallest currently available. First, we construct a candidate set for each row strategy containing only that strategy. We insert the sets into a priority queue where sets containing the fewest strategies receive highest priority. We repeatedly pop the smallest candidate set from the queue and add all the rationalizable strategies to that set using `all_rationalizable`. If new strategies were added, the resulting set is inserted back into the queue, and prioritized based on its new size. The algorithm terminates when a candidate set is removed from the queue that fails to admit any new rationalizable strategies. That set is returned, and it is a smallest minimal CURB set. We call this algorithm `small_MC`.

The smallest CURB set in any game must be minimal, and all other candidates in the queue are of size greater or equal to the returned set. The complexity of `small_MC` is bounded by the total number of strategies in the smallest CURB set, which we denote by n_{sc} . Since each call to `all_rationalizable` must add at least one strategy until `small_MC` terminates, we have the following.

Proposition 5. *Function `small_MC` returns a minimal CURB set that is a smallest minimal CURB set in the game, and it is $O(n_{sc} n^2) \times LFP(n)$.*

Proof. At the time `small_MC` terminates, `all_rationalizable` has been called on all row and column strategies in the set, with no new rationalizable strategies being discovered.

Therefore, the returned set is CURB. Since all other candidate sets on the queue must be as large, or larger than the returned set, (and future exploration can only add strategies to these sets) this set is at least as small as the smallest CURB set in the game. The smallest CURB set must also be a minimal CURB set.

Whenever a candidate set is fathomed, at least one new strategy must be added or `small_MC` will terminate. Since there are n candidate sets, and n_{sc} strategies in returned set, at most $n \times n_{sc}$ sets have been fathomed at termination. Each examination of a candidate set involves a call to `all_rationalizable` the complexity of `small_MC` is as given. \square

3.4 Experimental Results

We implemented the algorithms above and conducted experiments on their performance on all of the instance generators of the main (and perhaps only) benchmark collection for solving normal-form games, *GAMUT* [15]. The *GAMUT* collection includes a variety of game types from the game theory literature as well as games specifically designed to test different aspects of scalability of algorithms for solving games. The *GAMUT* instance generators allow one to generate multiple game instances of a given size, as well as studying the scalability of algorithms by generating instances of increasing size.¹ In this section we show that the complexity of our algorithms depends primarily on the size of the game and the size of its smallest CURB set. We then proceed to explore the distribution of CURB set sizes in the different games.

We first report on the run-time performance of our algorithms on two *GAMUT* game distributions: *random games*, and *covariant games*. Figure 1 shows how each of our minimal CURB set finding algorithms scales with game size on a data set of over 1000 random, square normal form games with total number of strategies n between 20 and 100. The experiments show that `small_MC` is faster than `all_MC`. This is consistent with their O -complexities, considering that many random games have small CURB sets. While the O -complexity of `one_MC` and `all_MC` is the same, experimentally `one_MC` is faster because it only needs to find one minimal CURB set. (On any game with more than one minimal CURB set, `one_MC` is faster than `all_MC`.)

We observed that the performance on random games illustrated in Figure 1 was typical of that of many other *GAMUT* instance distributions as well. However, to show potentially differing performance, we also report on experiments with the covariant game class, in which payoffs for both players are drawn from the same distribution with a specified covariance. In our experiments we used a covariance parameter of -0.5 . That class and that setting have been shown to be particularly challenging for Nash equilibrium finding (with the Lemke-Howson algorithm and the Porter-Nudelman-Shoham algorithm) [17]. Figure 1 shows that the `all_MC` algorithm scales similarly on random and covariant games, while the other two algorithms lose their speed advantages when applied to the covariant class.

Most random games have small smallest CURB sets (in fact, often sets of size 2, i.e.,

¹We did not benchmark on the *GAMUT* games that only have a fixed size, such as Chicken, Prisoner’s Dilemma, and Battle of the Sexes, because they are trivial to solve from a computational perspective.

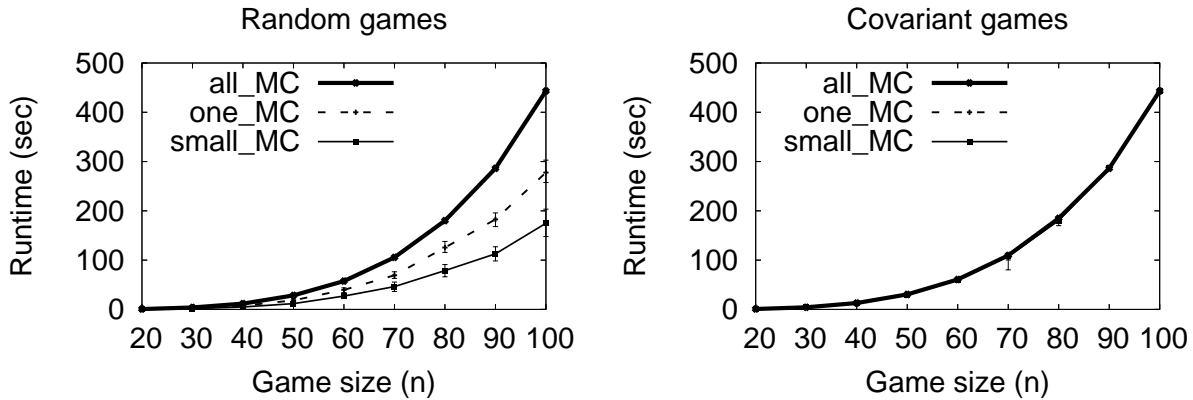


Figure 1: Scalability of our algorithms in game size.

pure-strategy equilibria), and those that do not, tend to have very large smallest CURB sets (Figure 2). On the other hand, covariant games tend to have almost no small smallest CURB sets and often have large smallest CURB sets. (This is consistent with the observed hardness of these games for support enumeration-based Nash equilibrium finding algorithms that try to find equilibria with small supports first [17].) The disparity explains the lowered performance on covariant games for both the two minimal CURB finding algorithms which are affected by the size of the smallest minimal CURB set.

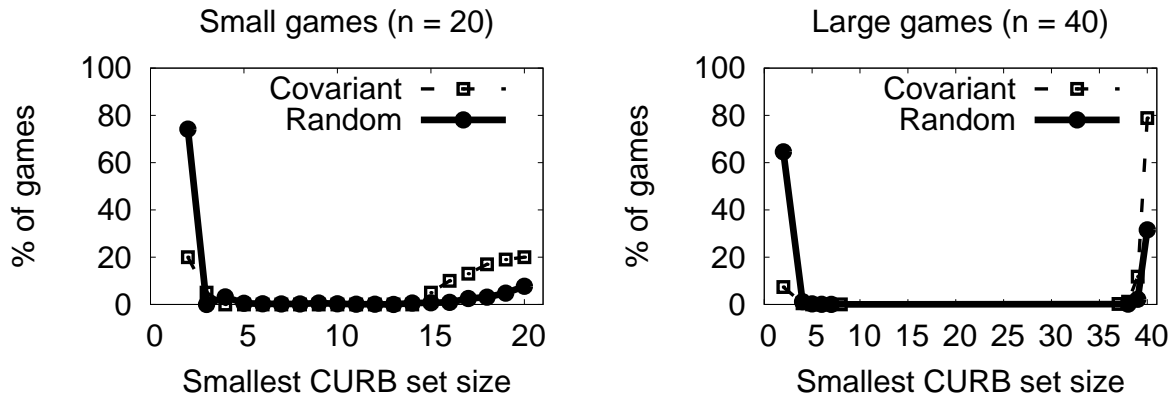


Figure 2: Distribution of smallest CURB set size in random and covariant ($r = -0.5$) games, where $n = 20$ and $n = 40$ (3,000 games for each distribution and value of n).

Figure 3 shows the distribution of smallest CURB set size for 1000 instances from each of the twenty-four other distributions emitted by *GAMUT* generators. This is becoming the standard way of testing game-solving algorithms [17, 19], and we used the same parameter settings in the distributions as those prior papers. For covariant games, the suffixes “Pos”, “Rand”, and “Zero” refer to positive, random, and zero covariance parameters respectively.

For distributions that take a graph as input, “CG”, “RG”, and “SG” refer to complete, random, and star graphs.

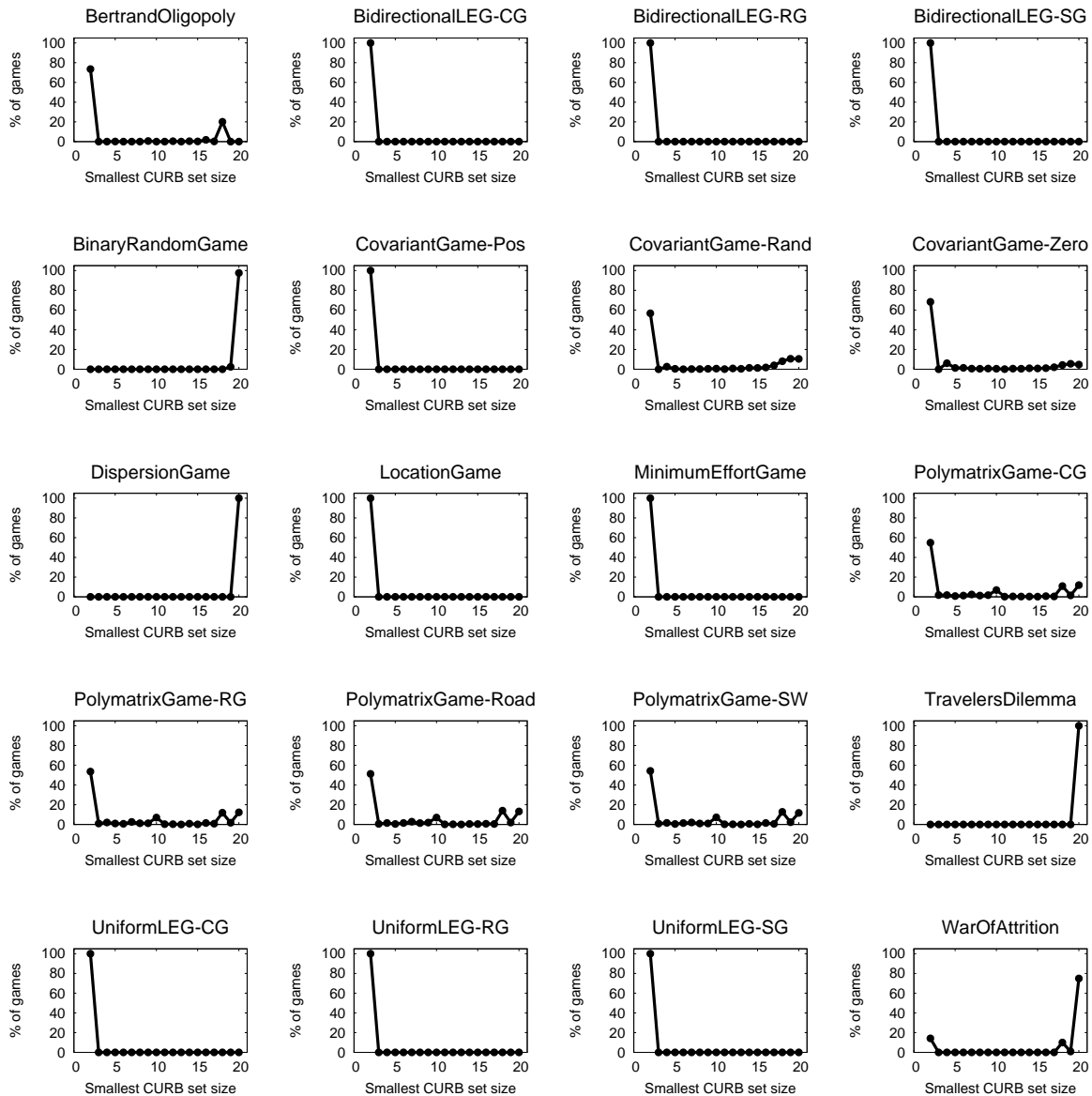


Figure 3: Distribution of smallest CURB set size in sets of 1000 games with $n = 20$ from various *GAMUT* distributions.

All of these distributions, like random and covariant games, exhibited very few medium-sized smallest CURB sets. Most of the instances have a smallest CURB set that is extreme: either a pure strategy equilibrium or the entire game. With some of the generators, all of the instances lie at the same extreme. Interestingly, some generators (e.g., Polymatrix) produced a significant number of games with CURB sets of one or more specific non-extremal sizes.

It is also notable that using different graph parameters for Local Effect and Polymatrix had no effect on their smallest CURB set size distributions.

To better understand how the minimal CURB set finding algorithms scale with the size of the smallest CURB set, we bucketed the $n = 20$ random and covariant games according to the size of the smallest CURB set. (For $n = 40$, the buckets for medium-sized smallest CURB sets were nearly empty, making it impossible for us to estimate mean run times with meaningful accuracy.) Figure 4 plots the average run time for each bucket. On games with very small CURB sets, `small_MC` is fastest, but it is outperformed by both `one_MC` and `all_MC` as the smallest CURB set grows. The surprising average-case efficiency of the latter two algorithms is due to their leveraging of information across calls to `min_containing_CURB` with different seeds. Because `small_MC` performs all the searches in parallel, this information is unavailable.

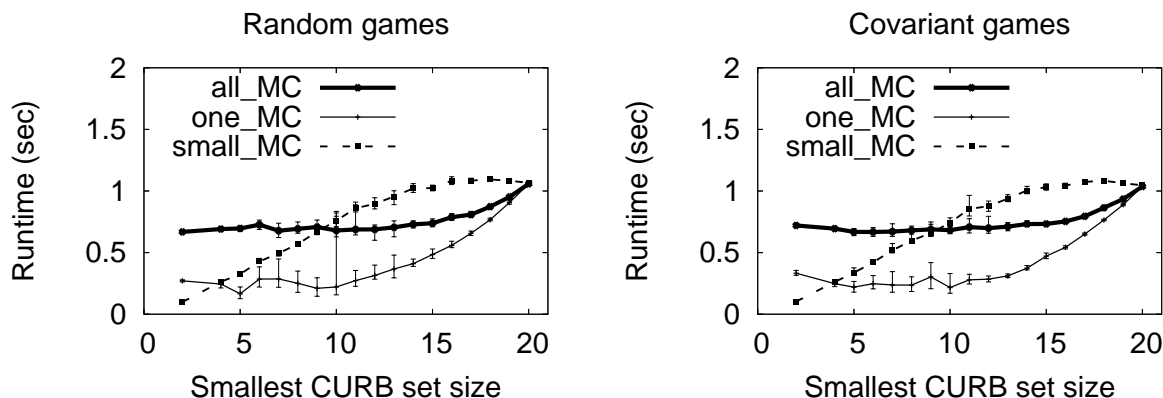


Figure 4: Average run time on games where $n = 20$, with varying smallest CURB set sizes.

4 CURB Sets and Nash Equilibria

Minimal CURB sets and Nash equilibria both model strategy subspaces which are mutually reinforced given the rationality of agents and their common knowledge. A CURB set with only one row and one column strategy is *necessarily* a pure-strategy Nash equilibrium. Furthermore, any minimal CURB set contains the supports for at least one Nash equilibrium in mixed strategies [2]. We observe that this result suggests a secondary use for finding minimal CURB sets: our algorithms can be used to preprocess a game so that a Nash equilibrium finding algorithm needs to only operate on a minimal CURB set rather than the entire game. This can yield an arbitrarily large reduction of the search space, as Theorem 3(a) will show.

The most common prior preprocessing technique for Nash equilibrium finding, iterated removal of dominated strategies, attempts to eliminate strategies that cannot be played with any probability in any Nash equilibrium [12, 9]. The same is true of a recent preprocessing technique, the generalized eliminability method [7]. One comparative advantage of minimal

CURB set-based elimination is that it can eliminate strategies that are played in some equilibria, while guaranteeing that the resulting set still contains the supports of at least one equilibrium.

Our CURB set-based preprocessor can reduce search space size by an arbitrary amount even on games where no prior preprocessing technique can eliminate anything:

Theorem 3. *For any $r \geq 2$ and $c \geq 2$, there exists normal form games of size $r \times c$, with the following properties:*

- a) *a minimal CURB set is contained with shape 1×1 or $r' \times c'$ (for any r', c' s.t. $1 < r' \leq r$ and $1 < c' \leq c$),*
- b) *iterated elimination of dominated strategies (even domination by mixed strategies) cannot eliminate any strategies,*
- c) *the recent recursive preprocessing technique (that **can** eliminate strategies that belong to some equilibrium as long as some other equilibrium remains) [8] cannot eliminate any strategies, and*
- d) *the general eliminability method [7] cannot eliminate any strategies.*

Proof. We first present a family, Γ , of games. Let $\Gamma_{r'c'}$ denote such a game of size $r' \times c'$. The following generator produces such a game where $r', c' \geq 2$. Assign the payoffs $u(s_{r_1}, s_{c_1}) = u(s_{r_2}, s_{c_2}) = (0, 1)$ and $u(s_{r_1}, s_{c_2}) = u(s_{r_2}, s_{c_1}) = (1, 0)$. Then, for $i \in [2, r' - 1]$, set $u(s_{r_{i+1}}, s_{c_1}) = (\frac{r'-i}{r'}, 1)$ and $u(s_{r_{i+1}}, s_{c_2}) = (\frac{i}{r'}, 0)$. Next, for $j \in [2, c' - 1]$, set $u(s_{r_1}, s_{c_{j+1}}) = (0, \frac{c'-j}{c'})$ and $u(s_{r_2}, s_{c_{j+1}}) = (1, \frac{j}{c'})$. For all payoffs still unassigned, set $u(s_{r_i}, s_{c_j}) = (-i, -j)$.

For example, the game $\Gamma_{3,4}$ is as follows.

$\Gamma_{3,4}$	s_{c_1}	s_{c_2}	s_{c_3}	s_{c_4}
s_{r_1}	0,1	1,0	$0, \frac{1}{2}$	$0, \frac{1}{4}$
s_{r_2}	1,0	0,1	$1, \frac{1}{2}$	$1, \frac{3}{4}$
s_{r_3}	$\frac{1}{3}, 1$	$\frac{2}{3}, 0$	-3,-3	-3,-4

Any game generated in this way has a Nash equilibrium where the row player mixes between his first two strategies and the column player mixes among all his strategies. It also has an equilibrium where the column player mixes between his first two strategies and the row player mixes among all his. Thus, every strategy in $\Gamma_{r'c'}$ is part of some equilibrium. Additionally each column strategy is a best response to a mixture over the first two row strategies (and, to any column strategy, one of those two is a best response), and vice versa. Thus, $\Gamma_{r'c'}$ has a single minimal CURB set and it includes the entire game.

We now construct an $r \times c$ game with a minimally CURB $r' \times c'$ subset by putting the game $\Gamma_{r'c'}$ in the top left and the game $\Gamma_{(r-r')(c-c')}$ in the bottom right. All other payoffs are set to negative random noise. The resulting game is shown in Figure 5.

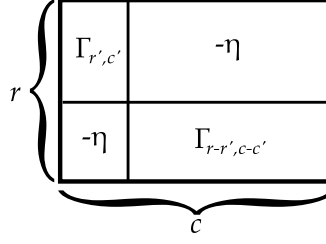


Figure 5: $r \times c$ game with arbitrary reduction to an $r' \times c'$ CURB set, irreducible by prior techniques.

It is irreducible by (iterated) dominance and by general eliminability because every strategy participates in some Nash equilibrium. The game is irreducible by the recursive preprocessor because the row player's payoffs are distinct within each column and the column player's payoffs are distinct within each row. \square

However, three factors curtail the promise of minimal CURB set algorithms as powerful preprocessors for Nash equilibrium finding. First, the fastest Nash equilibrium finding algorithms, while requiring exponential time in the worst case, tend to run faster than (at least the current implementations of) the CURB set finding algorithms on many instance distributions. Second, by Theorem 3(a), the smallest CURB set can be arbitrarily large (up to the size of the entire game, in which case the preprocessor does not eliminate any strategies from consideration). Third, even after the smallest minimal CURB set has been identified, the remaining search space (CURB set size) can be arbitrarily larger than the size of the supports of a contained Nash equilibrium:

Theorem 4. *A Nash equilibrium with supports consisting of two strategies for each player can be the only Nash equilibrium in an arbitrarily large minimal CURB set.*

Proof. Consider the following family of games that contain large minimal CURB sets and small-support equilibria. For any integer $k > 0$, we define the game Ω_k as follows. As in the previous proof, assign the payoffs $u(s_{r_1}, s_{c_1}) = u(s_{r_2}, s_{c_2}) = (0, 1)$ and $u(s_{r_1}, s_{c_2}) = u(s_{r_2}, s_{c_1}) = (1, 0)$. Then, for $i \in [3, 2 + k]$,

- $u(s_{r_i}, s_{c_1}) = (-\infty, \epsilon)$, $u(s_{r_1}, s_{c_i}) = (\epsilon, -\infty)$,
- $u(s_{r_i}, s_{c_i}) = (0, 0)$,
- $u(s_{r_i}, s_{c_{i-1}}) = (1 + \epsilon, 0)$, $u(s_{r_{i-1}}, s_{c_i}) = (0, 1 + \epsilon)$, and
- for all $j > i + 1$ and $j \leq 2 + n$,
 $u(s_{r_i}, s_{c_j}) = (0, -\infty)$, and $u(s_{r_j}, s_{c_i}) = (-\infty, 0)$

For example, the game Ω_2 is as follows.

Ω_2	s_{c_1}	s_{c_2}	s_{c_3}	s_{c_4}
s_{r_1}	0,1	1,0	$\epsilon, -\infty$	$\epsilon, -\infty$
s_{r_2}	1,0	0,1	$0, 1+\epsilon$	$0, -\infty$
s_{r_3}	$-\infty, \epsilon$	$1+\epsilon, 0$	0,0	$0, 1+\epsilon$
s_{r_4}	$-\infty, \epsilon$	$-\infty, 0$	$1+\epsilon, 0$	0,0

Proposition 6. Ω_k has a single minimal CURB set and it includes the entire game.

Proof. Strategies $s_{r_1}, s_{r_2}, s_{c_1}$, and s_{c_2} must be included in some minimal CURB set, as they each form best responses to each other in the subgame containing them, and this subgame admits no pure-strategy Nash equilibrium. To prove that all other strategies in the game are also in the minimal CURB set, we introduce the following Lemma.

Lemma 1. For $i > 2$, the row (column) player's strategy s_{r_i} (s_{c_i}) is a best response to the column (row) player's strategy $s_{c_{i-1}}$ ($s_{r_{i-1}}$). The best response to the row (column) player's last strategy $s_{r_{n+2}}$ ($s_{c_{n+2}}$) is s_{c_1} (s_{r_1}).

Lemma 1 holds based on the construction of the Ω game: the strategies in question provide ϵ more utility than any others. Based on this Lemma, we can see that when $i = 3$, the row (column) player's strategy s_{r_3} (s_{c_3}) is a best response to the column (row) player's second strategy. This forces the third strategy of each player into the minimal CURB set containing the first two strategies of each player, and inductively each additional strategy is added in the same way. \square

Proposition 7. In Ω_k , the only Nash equilibrium is the the strategy profile where $s_{r_1}, s_{r_2}, s_{c_1}$ and s_{c_2} are each played with probability $\frac{1}{2}$.

Proof. Assume, for contradiction, that this is not the case, that is, there exists a mixture, m_r^* , over the rows M_r^* , comprising the row player's profile in a Nash equilibrium, and $s_{r_1} \notin M_r^*$. Along with our assumption, the definition of Nash equilibrium implies that there must exist a mixture, m_c^* , over columns M_c^* such that $\beta_r(m_c^*) = M_r^*$ and $\beta_c(m_r^*) = M_c^*$. Since s_{r_1} is not in M_r^* by assumption, there exists $i > 1$ such that s_{r_i} is the lowest numbered support in M_r^* , and the definition of Ω specifies the outcome, $u(s_{r_i}, s_{c_j}) = (0, -\infty)$, when $j > i + 1$.

The column player's Nash equilibrium supports cannot contain any such s_{c_j} , because placing any positive probability on this strategy will lead to an expected payoff of $-\infty$ and playing the pure strategy s_{c_1} provides guaranteed payout of at least 0. If we exclude these strategies, $s_{c_{i+1}}$ (the highest-numbered remaining column strategy) is the only remaining strategy, other than s_{c_1} , which provides non-zero utility against mixtures on rows $\geq i$. In other words, it dominates all column strategies on the row player's supports M_r^* , aside from one: s_{c_1} .

Since dominated strategies cannot be played in equilibrium, M_c^* is constrained to a subset of $\{s_{c_1}, s_{c_{i+1}}\}$. If M_c^* contains s_{c_1} , M_r^* must not include any s_{r_j} with $j > 2$, due to the $-\infty$ expected payoff of any mixture including such strategies (as discussed above). In this case, the only remaining possible row Nash equilibrium mixture is the pure strategy s_{r_2} , the best

response to which is s_{c_3} . Since, by Corollary 2, Ω^n has no pure Nash equilibrium, this cannot constitute an equilibrium, contradicting our assumption. Alternatively, if M_c^* does not include s_{c_1} , then m_c^* must be the pure strategy $s_{c_{i+1}}$, and Lemma 2 provides a pure-strategy best response to any pure strategy with $i > 2$. This would, again, form a pure strategy Nash equilibrium, which we have shown does not exist.

The reasoning in this proof can also be inverted to show that a contradiction is caused by the assumption that M_c^* does not contain s_{c_1} . \square

This completes the proof of Theorem 4. \square

The above results mean that minimal CURB set algorithms do not always help in Nash equilibrium finding because the smallest CURB set can be arbitrarily large (Theorem 3(a)) and it can be arbitrarily loose around an enclosed Nash equilibrium (Theorem 4). However, on any game instance that has a small CURB set *or* a relatively tight minimal CURB set², and on which standard equilibrium-finding algorithms run very slowly, our technique will yield a drastic speed improvement for Nash equilibrium finding (the worst-case complexity of our preprocessor is polynomial, while any equilibrium-finding algorithm has super-polynomial run-time (unless PPAD=P)).

Furthermore, the existence of the polynomial-time algorithm for detecting a game’s smallest CURB set (`small_MC`) allows us to offer the following theoretical result of potential general interest.

Theorem 5. *The complexity of finding a Nash equilibrium for a two-player normal-form game can be super-polynomial only in the size of the game’s smallest CURB set.*

Proof. This is clear given the existence of the polynomial-time `small_MC` algorithm for finding the smallest minimal CURB set, and the fact that every minimal CURB set contains the supports for at least one Nash equilibrium [2]. \square

The relationship between the complexity of finding a minimal CURB set and that of finding a Nash equilibrium is surprising in several ways. For one, it is not obvious that finding a minimal CURB set should be easier than finding a Nash equilibrium, since, like Nash equilibria, CURB sets have an exponential space of possible supports which are chosen through maximization processes for both players. In fact, being a set-valued concept, finding minimal CURB sets may seem intuitively more complex. Yet from a theoretical worst-case perspective, Nash equilibrium finding is PPAD-complete (which is widely believed to be a strictly harder complexity class than P) and, as we showed earlier in this paper, minimal CURB set finding is polynomial-time.

What about the complexity of the two problems in practice? As Figure 6 shows, the run times of our smallest CURB set finding algorithm and the Lemke-Howson Nash equilibrium finding algorithm (using the implementation in *GAMBIT* [14]) seem to scale similarly with

²If the game has a relatively tight CURB set, a Nash equilibrium can be found quickly by enumerating strategies of the CURB to be left out from the supports.

input game size (when one does not explicitly generate those pathological cases which produce exponential behavior in the latter [20]). In fact, the former is *slower* (by two orders of magnitude) on average than finding a Nash equilibrium. This experimental performance agrees with intuition but is the reverse of the theoretical state of knowledge about worst-case complexity.

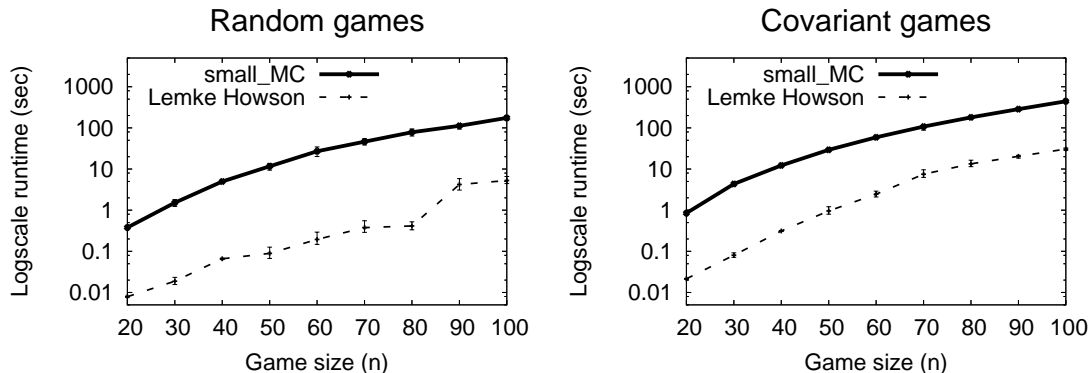


Figure 6: Average run time of `small_MC` and Lemke Howson as a function of game size.

The root cause of the complexity of Nash equilibrium search has proved elusive, as two candidates which were considered potential culprits have recently been shown not to affect worst-case complexity: games with two players and payoffs restricted to $\{0, 1\}$ are just as difficult as the general case, even if both restrictions apply simultaneously [5, 1]. The fact that bounding the smallest CURB set size *does* serve to bound the difficulty of Nash equilibrium search suggests that we can further isolate the cause of equilibrium search complexity as being endemic to minimal CURB sets, rather than games in general. In this regard we observe that the special two-player game used by Chen and Deng to show PPAD-completeness [5] is itself a single minimal CURB set, and remains such under Abbott *et. al's* [1] transformation to $\{0, 1\}$ -payoffs.

5 Conclusions

We presented the first algorithms for rationalizability and CURB sets, two important set-valued solution concepts, for normal-form games. Our algorithms find all rationalizable strategies for a given support of opponent's strategies, all minimal CURB sets (`all_MC`), one minimal CURB set (`one_MC`), and a smallest minimal CURB set (`small_MC`), all in polynomial time. The algorithms are based on basic properties of CURB sets which we prove, such as the fact that minimal CURB sets cannot overlap. The algorithms use dovetailing with a priority queue, and exploiting information across overlapping non-minimal CURB sets, to further improve speed.

Experiments on random games showed that `small_MC` is the fastest, `one_MC` is second, and `all_MC` is the slowest of the three. On covariant games, the speed advantage of the

former two disappears. The run time of the algorithms is primarily determined by the size of the smallest CURB set. On games with small CURB sets, `small_MC` is significantly faster than the others, but the others (especially `one_MC`) are faster on games containing only large CURB sets.

Our algorithms also enable the study of CURB set size distributions of different game classes. We showed that the instance distributions from *GAMUT* are mainly extremal in the sense that a given game generator will yield mostly games with pure-strategy equilibria and/or games where the game itself is the single minimal CURB set. However, curiously, some of the generators yield a significant number of games with smallest CURB sets of specific non-extremal sizes.

We also examined the potential for using our algorithms as preprocessors for Nash equilibrium finding algorithms. We proved that our technique can eliminate an arbitrarily large portion of the game from consideration while guaranteeing that the remaining strategies contain a Nash equilibrium. This is the case even on games where prior preprocessing techniques are powerless.

On the downside, we showed that the smallest CURB set can be arbitrarily large and/or arbitrarily loose. Furthermore, on many distributions, we showed that current Nash equilibrium finding algorithms run faster on average than the CURB set algorithms. This is surprising in that the theoretical worst-case complexity of the two problems is the reverse.

We showed that the worst-case complexity of finding a Nash equilibrium is polynomial in all aspects of the game *except* the size of the smallest CURB set. Taken together with our CURB set finding algorithms that are polynomial even in the worst case, and the fact that Nash equilibrium finding is super-polynomial in the worst case (unless $\text{PPAD}=\text{P}$), we see that the essence of the worst-case complexity of finding a Nash equilibrium is the complexity of finding a Nash equilibrium within a minimal CURB set.

Our CURB set finding algorithms (in particular `small_MC`) can be used as a preprocessing technique for finding a Nash equilibrium. Since the preprocessing technique is of polynomial complexity, this approach will yield a drastic speed improvement on hard games that have a small *or* a relatively tight minimal CURB set.

The CURB set definition is for any number of players. In this paper we focused on the two-player case. For a larger number of players, the only obstacle to finding minimal CURB sets is finding rationalizable strategies quickly as a subroutine. The mathematical program we use is of degree $d-1$ where d is the number of players; for two players it is a linear feasibility problem but for three players the constraints are already quadratic. Our CURB set finding algorithms can be trivially generalized to any number of players. Even the generalized versions only make a polynomial number of calls to the `all_rationalizable` subroutine. Therefore, if future research finds polynomial algorithms for finding rationalizable strategies for more than two players in polynomial time, then our generalized CURB set algorithms are also polynomial time.

References

- [1] Tim Abbott, Daniel Kane, and Paul Valiant. On the complexity of two-player win-lose games. In *Proceedings of the Foundations of Computer Science (FOCS)*, 2005.
- [2] Kaushik Basu and Jorgen W. Weibull. Strategy subsets closed under rational behavior. *Economics Letters*, 36(2):141–146, 1991.
- [3] Pierpaolo Battigalli and Marciano Siniscalchi. Rationalizable bidding in first-price auctions. *Games and Economic Behavior*, 45(1):38–72, 2003.
- [4] B Douglas Bernheim. Rationalizable strategic behavior. *Econometrica*, 52(4):1007–28, 1984.
- [5] Xi Chen and Xiaotie Deng. Settling the complexity of 2-player Nash-equilibrium. Technical Report TR05-140, Electronic Colloquium on Computational Complexity, 2005.
- [6] Vincent Conitzer and Tuomas Sandholm. Complexity of (iterated) dominance. In *Proceedings of the ACM Conference on Electronic Commerce (ACM EC)*, pages 88–97, 2005.
- [7] Vincent Conitzer and Tuomas Sandholm. A generalized strategy eliminability criterion and computational methods for applying it. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 483–488, 2005.
- [8] Vincent Conitzer and Tuomas Sandholm. A technique for reducing normal form games to compute a Nash equilibrium. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Hakodate, Japan, 2006.
- [9] I. Gilboa, E. Kalai, and E. Zemel. The complexity of eliminating dominated strategies. *Mathematics of Operations Research*, 18:553–565, 1993.
- [10] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.
- [11] S. Hurkens. Learning by forgetful players. *Games and Economic Behavior*, 11(1):304–329, 1995.
- [12] D. E. Knuth, C. H. Papadimitriou, and J. N. Tsitsiklis. A note on strategy elimination in bimatrix games. *OR Letters*, 7(3):103–107, 1988.
- [13] C. Lemke and J Howson. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics*, 12:413–423, 1964.
- [14] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory, version 0.97.1.5, 2004.

- [15] Eugene Nudelman, Jennifer Wortman, Yoav Shoham, and Kevin Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of International Conference on Automated Agents and Multi-Agent Systems (AAMAS)*, pages 880–887, 2004.
- [16] David G Pearce. Rationalizable strategic behavior and the problem of perfection. *Econometrica*, 52(4):1029–50, 1984.
- [17] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 664–669, San Jose, CA, USA, 2004.
- [18] Vitaly Pruzhansky. On finding CURB sets in extensive games. *International Journal of Game Theory*, 32(2):205–210, 2003.
- [19] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 495–501, Pittsburgh, PA, USA, 2005.
- [20] Rahul Savani and Bernhard von Stengel. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2004.
- [21] Mark Voorneveld, Willemien Kets, and Henk Norde. An axiomatization of minimal CURB sets. *International Journal of Game Theory*, 33:479–490, 2005.
- [22] Yinyu Ye. Improved complexity results on solving real-number linear feasibility problems. *Mathematical Programming*, 106(2):339–363, 2006.