Computing Nash Equilibria of Action-Graph Games via Support Enumeration

David R. M. Thompson[1], Samantha Leung[2], and Kevin Leyton-Brown[1]

[1] {daveth, kevinlb}@cs.ubc.ca, University of British Columbia
[2] samlyy@cs.cornell.edu, Cornell University

Abstract. The support-enumeration method (SEM) for computation of Nash equilibrium has been shown to achieve state-of-the-art empirical performance on normal-form games. Action-graph games (AGGs) are exponentially smaller than the normal form on many important classes of games. We show how SEM can be extended to the AGG representation, yielding an exponential improvement in worst-case runtime. Empirically, we demonstrate that our AGG-optimized SEM algorithm substantially outperforms the original SEM, and also outperforms state-of-the-art AGG-optimized algorithms on most problem distributions.

1 Introduction

The canonical representation of simultaneous-move, perfect information games is the normal form. Because this representation grows exponentially in the number of players, it is impractical for modeling interactions that involve more than a handful of players. There is an extensive literature on compactly representing interesting games [19, 9, 12]. Action-graph games [8] (AGGs) unify these past representations and can furthermore represent many additional families of games in polynomial space. Computing solution concepts (e.g., Nash equilibrium) of compactly represented games is an active area of research. Although in some cases novel algorithms have been proposed [4, 20], a particularly fruitful approach has been to take existing, normal-form-based algorithms, and modify them to operate efficiently with a compact representation [2, 1]. Indeed, the state-of-the-art Nash-equilibrium-finding algorithms for AGGs are normal-form algorithms with AGG-optimized expected-utility calculations [8].

One algorithm that has not been modified to work with any compact representation also has very strong empirical performance: the support enumeration method (SEM) [18]. This algorithm also has other useful properties, such as returning the equilibrium in which agents randomize as little as possible, and being able to find all equilibria. A key reason that SEM has not been extended to work with compact game representations is that it operates very differently from other equilibrium-finding algorithms, and hence existing techniques cannot be applied to it directly. In this paper we show how SEM can be extended to work with AGGs (and hence with other game families compactly encodable as AGGs, such as graphical games and congestion games). Specifically, we show how three of SEM's subroutines can be made exponentially faster. Experimentally, we show that these optimizations dramatically improve SEM's performance, rendering it competitive with and often stronger than other state-of-the-art algorithms for computing equilibria of AGGs.

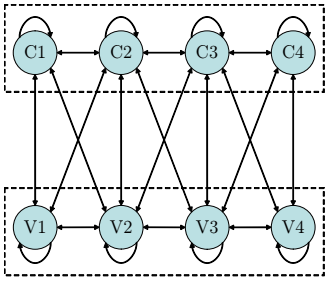

Fig. 1. An action graph for the "ice cream game" [8]. In the ice cream game, each player has either chocolate (C) or vanilla (V) ice cream to sell and must choose one of four possible locations to set up his stand. The action-graph encodes that a player's payoff depends only on his location, and the type and number of competitors within one step of his location.

2 Technical Background

In this section we summarize the two strands of related work that this paper brings together: AGGs and SEM.

2.1 Action-Graph Games

Action-graph games [8] achieve compactness by exploiting several kinds of structure in the payoffs. The first kind of structure is anonymity, which means that an agent's payoff depends only on his own action and the "configuration" induced by the other agents–a tuple of counts of how many agents played each action. Anonymity means that an agent's payoff does not depend on who played which action. This yields representational savings because, instead of storing a payoff for every pure-strategy profile, AGGs only need to store one for every configuration. The second kind of structure is context-specific independence, which means that a given agent only cares about the actions of others who take *a specific subset* of their actions. (This is a strengthening of (strict) independence, as captured by Graphical Games [9], which says that the agent never cares about certain others' actions, regardless of which actions they choose.) The context-specific aspect is that the actions (and hence agents) that can affect a given agent's payoff depend on which action that agent chooses. These independencies are encoded in an action graph, a directed graph where each node corresponds to an action, and payoff for playing an action only depends on the counts on its neighboring nodes. Instead of storing an action's payoff for every configuration, AGGs only need to store one for every "projected configuration," a tuple of counts on the neighbors of a vertex (denoted $C^{(v)}$ for vertex v). See Figure 1 for an example.

Formally, an action-graph game is a 4-tuple (N, A, G, u) where N is the set of agents $(1, 2, ..., n)$; $A = \Pi_{i \in N} A_i$ is the set of action profiles; $G = (V, E)$ is a directed graph with vertices V (where $V = \bigcup_{i \in N} A_i$) and edges E; and $u = (u_1, u_2, ..., u_{|V|})$ is a tuple of utility functions, where $u_v : C^{(v)} \mapsto \mathbb{R}$.

$$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) = v_i \qquad \forall i \in N, \forall a_i \in S_i \tag{1}$$

$$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) \leq v_i \qquad \forall i \in N, \forall a_i \in A_i \setminus S_i \tag{2}$$

$$\sum_{a_i \in S_i} p_i(a_i) = 1 \qquad \forall i \in N \tag{3}$$

$$p_i(a_i) \geq 0 \qquad \forall i \in N, \forall a_i \in S_i \tag{4}$$

Fig. 2. The Test-Given-Support (TGS) feasibility program for n-player games. For any given support profile $S \in \prod_{i \in N} 2^{A_i}$, we can construct a TGS feasibility program where any feasible solution p, v is a Nash equilibrium with support S, where the players randomize according to the probabilities in p and get the payoffs specified by v. The constraints on line (1) specify that each player is indifferent between all the actions in his support. Those on line (2) specify that each player weakly prefers the actions in his support. The remaining lines specify that each mixed strategy is a probability distribution. (Note that this formulation allows for actions in the support to be played with zero probability. This doesn't adversely affect SEM's behavior; if such a Nash equilibrium existed, SEM would have found it already.)

Note that there is a more complicated family of AGGs, AGGs with function nodes (described in detail in [8]). The algorithms described in this paper also work (and have the same asymptotic performance) with function nodes, but we omit the description of these games for notational clarity and simplicity.

The other advantage of AGGs, besides their ability to represent games compactly, is that they can be reasoned about efficiently. In particular, given a mixed-strategy profile it is possible in polynomial time to compute a given agent i's expected utility for playing action a_i, using dynamic programming [8]. The dynamic program proceeds through n iterations, where at the k^{th} iteration, it computes the marginal distribution over the projected configurations $C^{(a_i)}$ given the strategies of the first k agents.

2.2 Support-Enumeration Method

The support-enumeration method (SEM) is a brute-force-search method of finding Nash equilibria. However, rather than searching through all mixed strategy profiles, it searches through support profiles (specifying which actions each agent plays with positive probability) and tests whether there is a Nash equilibrium with that particular support. This test can be performed using the polynomial feasibility program given in Figure 2. Though several algorithms have been proposed for searching in the space of supports to find Nash equilibria [14, 5, 13], we will focus on the most recent SEM variant, due to Porter, Nudelman and Shoham [18]. This variant introduces two important features designed to improve empirical performance. First, it uses heuristics to order its exploration of the space of supports, searching from smallest to largest, breaking ties in favor of more balanced support profiles. This order can speed up equilibrium finding for several reasons. First, there are fewer small support-size profiles to search through. Second, the corresponding feasibility programs have fewer variables and smaller constraints. And third, in games of interest (see, e.g., [11]), Nash equilibria with small,

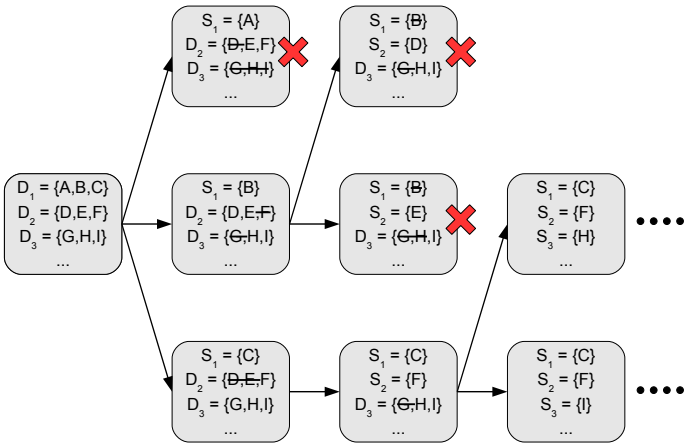

Fig. 3. Porter *et al*'s [18] tree-search works by instantiating strategies from agents' supports, one agent at a time, and removing strategies that are strictly dominated conditional on the agents playing within their given supports. The search backtracks whenever an agent has an empty support or the TGS feasibility program is infeasible.

balanced supports are common [18]. Second, instead of simply iterating through the complete set of support profiles of a given size, SEM explores this space by tree search (see Figure 3 for an example). This search works by, at each level, selecting a support for a single additional player. At the leaves of the tree, the support is specified for every agent, and that support profile can be tested using TGS for the existence of a Nash equilibrium. The advantage of using search comes from pruning: after an agent's support is selected, SEM performs iterative removal of strictly dominated strategies (IRSDS), conditional on agents only playing actions in their supports. This has the effect of eliminating many support profiles from consideration. The search backtracks whenever an agent has an empty support or the TGS feasibility program is infeasible.

There are several reasons to be interested in the SEM algorithm. One is that it is the only known method for finding small-support equilibria. Another is that it has been shown to achieve better empirical performance than the previous state-of-the-art algorithms for the sample equilibrium problem, simplicial subdivision (SimpDiv) [10] and the global Newton method (GNM) [7], on many game families of interest. Notably, most of these games had pure-strategy Nash equilibria (PSNEs), which SEM finds in polynomial time. However, even on games without PSNEs—where SEM has exponential worst-case running time—SEM is still often faster than SimpDiv and GNM.

3 SEM for AGGs

Observe that we can trivially make a version of SEM that takes AGGs as input, simply replacing the normal form game (NFG) utility lookups ($u_i(a)$) with the AGG equivalents ($u_i(a) = u_{a_i}(c^{(a_i)})$, where $c^{(a_i)}$ is the projected configuration given a). We denote this

algorithm NFG-SEM, because its behavior is exactly the same as that of SEM for normal-form games. However, because AGGs can be exponentially smaller than NFGs, we show below that NFG-SEM's asymptotic worst-case performance, as a function of the length of its input, can be exponentially worse than that of SEM for the induced normal form of the same game. Specifically, we show that NFG-SEM's inner-loop operations—iterative removal of strictly dominated strategies and the TGS feasibility program—are at least worst-case exponential in the AGG input length (denoted ℓ). The outer-loop search over supports also requires exponential time, even for games with PSNEs.

However, we can do better if we construct a version of SEM that explicitly takes AGG structure into account. We present such an extension of SEM, denoted AGG-SEM, and its asymptotic analysis. Overall, we show that AGG-SEM's worst-case performance is exponentially faster than that of NFG-SEM.

3.1 Conditional Dominance

Because SEM makes extensive use of iterative removal of strictly dominated strategies, efficiently identifying dominated strategies is critical. For normal-form games, testing whether or not some pure strategy a_i is dominated by some other a'_i is straightforward: one can exhaustively search through through the pure strategy profiles of the other agents, looking for the existence of some a_{-i} to which a_i is a weakly better response. This trivial algorithm only requires time linear in the size of a normal-form game. However, it can require time exponential in the size of an action-graph game.

Lemma 1. *NFG-SEM's dominance check has a worst-case running time of* $\Theta(2^\ell)$.

Proof. Consider the family of action-graph games with two actions per player and no edges. For this family, there are at most $2n$ nodes, the payoff table for each of which contains only a single value. Thus, ℓ is $\Theta(n)$, while $|A_{-i}| = 2^{n-1}$ and therefore is $\Theta(2^\ell)$. In the worst case, exhaustive search iterates over every $a_{-i} \in A_{-i}$ to confirm that a_i is not a best response to any action profile. □

However, we can do better: a straightforward, polynomial-time algorithm for AGG dominance checking can be derived from Jiang et al's [8] dynamic-programming algorithm. To determine whether or not a_i is dominated by a'_i, we do not need to search through the entirety of A_{-i}; we only need to search over the set of possible projected configurations on the joint neighborhoods of a_i and a'_i. This adaptation guarantees polynomial runtime. However, empirically we observed that it often gave rise to poor performance, compared to exhaustive search over A_{-i}. We attribute this to stopping conditions: the exhaustive search can stop as soon as it finds any case where a_i is a better response, while the dynamic-programming algorithm must build up all configurations first before it ever encounters a better response, effectively performing a breadth-first search. Based on this insight, we created a depth-first-tree-search-based algorithm that combines the best of both approaches: like exhaustive search, it can find a better response without needing to compute the entire set of projected configurations; like our adaptation above, it exploits AGG structure and so needs only to evaluate a polynomial number of projected configurations. It works as follows. At each level, the search fixes the action of some agent, giving a search tree that potentially includes every A_{-i}. However, we

also perform multiple-path pruning: a search refinement in which previously visited nodes are recorded, and the search backtracks whenever it re-encounters a node along a different search path [17]. In our case, the algorithm backtracks whenever it encounters a previously visited projected configuration, based on a lookup from a trie map.

Lemma 2. *AGG-SEM's dominance check has a worst-case running time of* $O(nm\ell^3)$.

Proof. The search traverses a tree with a depth of n and a branching factor of m (where $m = \max_{i \in N} |A_i|$). However, at every level, at most ζ^2 nodes are expanded (where ζ denotes the largest set of possible projected configurations for any node), because there are at most ζ^2 distinct projected configurations on the neighborhood of a_i, a'_i. In the worst case, when it traverses the whole tree, the search must follow each of m arcs from $O(\zeta^2)$ nodes at each of n levels, or $O(nm\zeta^2)$ arcs. For each arc, the search may perform a trie-map lookup and insert; these operations each require runtime that grows like the maximum in-degree of the graph, ι, and so the total cost is $O(nm\zeta^2\iota)$. Because ℓ is $\Omega(\zeta + \iota)$, $nm\zeta^2\iota$ is $O(nm\ell^3)$. □

3.2 TGS Feasibility Program

SEM's asymptotic performance is dominated by the Test-Given-Support feasibility program. (Polynomial feasibility is NP-hard; e.g., polynomial constraints generalize 0–1 integrality constraints [22].) For NFG-SEM, this complexity obstacle is particularly severe: directly representing the TGS feasibility program requires space exponential in the size of the AGG. Thus, unless $P = NP$, TGS requires doubly exponential time.

Lemma 3. *The NFG-SEM TGS feasibility program has worst-case size of* $\Theta(nm2^\ell)$.

Proof sketch, similar to Lemma 1's proof. TGS can have $|A_{-i}|$ terms in each constraint, and this quantity is $\Theta(2^\ell)$ in the worst case. There are $O(nm)$ such constraints. □

The essential challenge is in the expected utility constraints (lines 1 and 2 of Figure 2) which can be exponentially long. We already have Jiang et al's [8] dynamic-programming algorithm for computing expected utility given a specific mixed-strategy profile. Now we want to compute expected utility symbolically, without specifying the probabilities beforehand. This can be accomplished by "unrolling" the dynamic program: every update in the dynamic program is expressed as a polynomial equality constraint in the TGS program. This set of new constraints is polynomial in the size of the AGG.

Lemma 4. *The AGG-SEM TGS feasibility program has worst-case size of* $O(n^2m^2\ell^2)$.

Proof. For each $j \in \{1, \ldots, n\}$ we introduce $O(\zeta)$ new constraints, each corresponding to the probability of a projected configuration given the strategies of the first j agents. This gives $O(n\zeta)$ constraints. Each contains at most $O(\zeta m)$ terms, corresponding to the possible projected configurations and actions that could lead to some new projected configuration when another agent is added. Since ζ is $O(\ell)$, the output requires $O(nm\ell^2)$ space. It must be run once for each agent i and for each action in A_i: $O(nm)$ times in total. Thus, the TGS feasibility program requires $O(n^2m^2\ell^2)$ space. □

Although this optimization speeds up the worst case exponentially, it is not guaranteed to be helpful on average. This is because the symbolic representation of the TGS system is made exponentially smaller by replacing each exponentially long expected-utility constraint with multiple small constraints. How this change affects runtime in the average case depends on the (black-box) feasibility solver.

3.3 Asymptotic Analysis of SEM for AGGs

We are now ready to demonstrate that asymptotically, AGG-SEM is exponentially faster than NFG-SEM. We assume that both algorithms make use of a polynomial feasibility solver with worst-case runtime $O(2^x)$ where x is the length of the feasibility program.

Theorem 1. *Assume that we have access to a polynomial feasibility solver with worst-case runtime $O(2^x)$, where x is the length of the feasibility program. Then NFG-SEM requires doubly exponential time time to find a sample Nash equilibrium in an AGG.*

Proof sketch. In the worst case, AGGs can have $\Omega(2^\ell)$ support profiles (as in Lemma 1), even for symmetric games. Thus, the search must traverse a tree with $O(2^\ell)$ leaf nodes where TGS is solved, and $O(2^\ell)$ interior nodes where iterative removal of strictly dominated strategies (IRSDS) is performed. Solving TGS takes $O(2^{nm2^\ell})$ runtime in the worst case. This expression is $O(2^x)$ where x is $O(nm2^\ell)$ by Lemma 3, and so dominates IRSDS. Thus the total runtime is $O(2^{nm2^\ell+\ell})$. □

Theorem 2. *AGG-SEM requires (only) exponential time to find a sample Nash equilibrium in an AGG.*

Proof sketch. AGG-SEM still searches $O(2^\ell)$ nodes, but TGS now requires $O(2^{n^2m^2\ell^2})$ time (Lemma 4), which again dominates IRSDS. The total runtime is $O(2^{n^2m^2\ell^2+\ell})$. □

Given complexity results known in the literature, it is unsurprising that AGG-SEM requires exponential time in the worst case. In particular, finding even PSNEs of AGGs in polynomial time would imply P=NP: AGGs generalize graphical games, and finding a PSNE of an arbitrary graphical game is NP-hard [6]. Further, finding a PSNE of a symmetric AGG with unbounded m is also known to be NP-hard [4].

3.4 Further Speedups for k-Symmetric Games

We now show that the search over supports can be sped up in the case of AGGs with k-symmetry, i.e., where the players can be partitioned into k classes such that all players in a class are identical. (We describe the algorithm for the case of $k = 1$, or full symmetry. The generalization is straightforward.) We saw in the proof of Theorem 1 that symmetry does not help for NFG-SEM. Here we strengthen that result, showing that NFG-SEM can take exponential time even when PSNEs exist in k-symmetric AGGs with bounded m and k.

Theorem 3. *NFG-SEM requires exponential time time to find a sample PSNE in a k-symmetric AGG with bounded m and k.*

Proof sketch. For games with PSNEs, we never need to solve TGS: any support profile that survives IRSDS is a Nash equilibrium. The tree search must still expand $O(2^\ell)$ nodes to explore all $O(2^\ell)$ pure support profiles. At each interior node, IRSDS is called, requiring $O(n^2m^3)$ calls to the conditional dominance test, which requires $O(2^\ell)$ time (by Lemma 1). For bounded m, the total runtime to find a PSNE is $O(n^2 2^{2\ell})$. □

Next, we show that we *can* achieve an improvement on such games for AGG-SEM. This optimization works by skipping any support profile that is a permutation of a previously explored support profile. At every stage of the tree search, we explore a support S_i iff $S_i \succeq S_j$ where j is any player with support selected higher in the tree, and where $\succ$ is the order in which supports are explored at each level of the tree.

Lemma 5. *AGG-SEM's search evaluates* $\mathrm{poly}(n)$ *support profiles in the worst case, even for games without PSNEs, given a* k*-symmetric AGG with bounded* k *and* m*.*

Proof. Every distinct support profile can be identified by a vector of $O(k2^m)$ integers in the range $[0, n]$, where each element indicates how many agents of a given class have a given support. There are at most $O(n^{k2^m})$ such vectors. For bounded k and m, this quantity is $\mathrm{poly}(n)$. □

Theorem 4. *AGG-SEM requires* $\mathrm{poly}(\ell)$ *time to find a sample PSNE in a* k*-symmetric AGG with bounded* m *and* k*.*

Proof sketch. For bounded m and k, AGG-SEM's search expands polynomially many nodes (by Lemma 5), each of which requires running IRSDS. IRSDS performs $O(n^2m^3)$ conditional dominance tests, requiring $O(nm\ell^3)$ time (by Lemma 2). Thus, AGG-SEM has $\mathrm{poly}(\ell)$ runtime on such games. □

4 Experimental Evaluation

So far, our analysis has concentrated on the worst case. However, improvements to the worst case do not necessarily imply improvements on instances of interest. As we are motivated by developing *practical* methods for computing Nash equilibria, we conducted an experimental evaluation to compare the performance of NFG-SEM and AGG-SEM.

4.1 Experimental Setup

We sampled 50 instances from each of 11 different game distributions (see Table 1). Nine distributions were from GAMUT [11]; each had $n = 10$ players, $m = 10$ actions per player, and action graphs with in-degree at most five. The remaining two distributions were over position auction games with $n = 10$ players and up to $m = 11$ actions per player (though weakly dominated actions, which occurred frequently, were omitted by the generator) [21]. On each game, we compared AGG-SEM to three other algorithms: NFG-SEM, and the two existing state-of-the-art Nash-equilibrium-finding algorithms: GNM, the global Newton method [7], and SimpDiv, simplicial subdivision [10], both using Gambit implementations [15] extended to work efficiently with AGGs by [8]. All algorithms were given error tolerance of 10^{-10}. For AGG-SEM and NFG-SEM, we used minos [16] to solve the TGS feasibility problems.

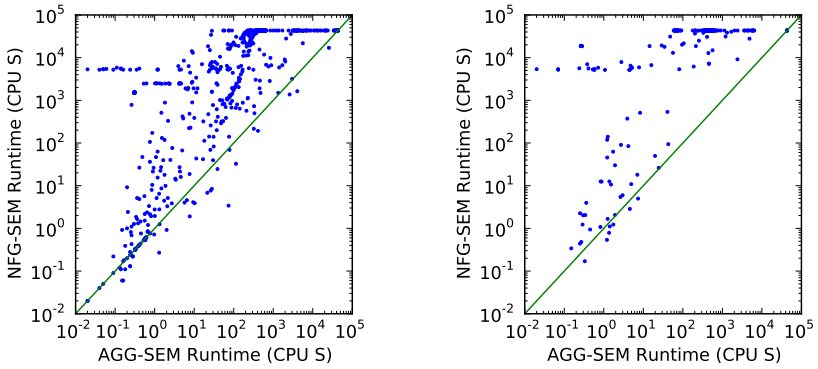

Fig. 4. Scatterplot contrasting the runtimes of AGG-SEM and NFG-SEM. The left plot shows runtimes for computing a sample Nash equilibria; the right plot shows runtimes for computing all pure strategy Nash equilibria.

We performed a blocking mean-of-means test [3] (with $p \leq 0.01$) to compare mean runtimes across game distributions. In three distributions (see Table 1), we were not able to conclude that differences were significant because of high runtime variation. Such problems can be overcome by obtaining additional data; thus, for these distributions we generated an additional 150 instances (i.e., 200 total). In the end, we were able to identify a significantly faster algorithm for every distribution.

Our experiments were performed on machines with dual Intel Xeon 3.2GHz CPUs, 2MB cache and 2GB RAM, running Suse Linux 11.1 (Linux kernel 2.6.27.48-0.3-pae). Each run was limited to 12 CPU hours; we report runs that did not complete as having taken 12 hours. In total, our experiments required about 420 CPU days.

4.2 Results

Overall, we found that AGG-SEM provided a substantial performance improvement over NFG-SEM, outperforming it on the vast majority of instances; see Figure 4. As we expected, AGG-SEM was not faster on absolutely every instance. Nevertheless, AGG-SEM achieved significantly faster mean performance in every game distribution. Its largest speedup over NFG-SEM was 280$\times$ (on D1), its smallest speedup was 1.45$\times$ (on D4), and its median speedup was 10$\times$ (on D10). While the biggest speedups were on games where AGG-SEM could leverage k-symmetry, ranging from 280$\times$ (on D1) to 7$\times$ (on D2), we still achieved substantial speedups on asymmetric games (those with n player classes), ranging from 10$\times$ (on D10) to 1.45$\times$ (on D4). AGG-SEM stochastically dominated NFG-SEM overall (see Figure 5), but on a per-distribution basis, it only stochastically dominated on four distributions (D3 and D9–11). AGG-SEM's failure to stochastically dominate on the remaining seven distributions was due to the fact that all contained instances that both methods solved extremely quickly (in less than one second), but that NFG-SEM finished more quickly. Considering runtimes over a second, AGG-SEM stochastically dominated NFG-SEM on every distribution.

AGG-SEM outperformed SimpDiv and GNM on 9 of the 11 game distributions (see Table 1), and furthermore stochastically dominated GNM overall (see Figure 5). Comparing the runtimes of AGG-SEM and SimpDiv, we found that the two were correlated: they both solved many (338) of the same instances in under 600s, with SimpDiv having better mean runtime on these instances ($\mu = 12.71$s vs $\mu = 108.32$s). On the remaining instances, SEM finished far more often (87.74% vs 23.11%). Thus, SimpDiv was only fastest on D2 (Ice-cream games), which contained almost exclusively instances that were easy for both algorithms. AGG-SEM only stochastically dominated SimpDiv on D10 and D11, which contained no instances that were easy for simpDiv. AGG-SEM's runtime was less correlated with that of GNM than it was with SimpDiv. For example, GNM solved every instance in D4 (GFP position auctions), which contained instances that were not solved by either SimpDiv or AGG-SEM. Overall, however, GNM had the worst performance (and was stochastically dominated by AGG-SEM on every distribution but D3 and D4). Although AGG-SEM had the fastest overall average runtime, there were at least a few instances for which each of AGG-SEM, SimpDiv and GNM was hundreds of times faster than the others. The best practical approach may thus be a portfolio of all three algorithms, following e.g. [23].

Like Porter, *et. al.*, [18], we found that in many (7 of 11) distributions, most games (over 90%) had PSNEs. AGG-SEM finished on every such game. Thus the four distributions (D4 and D9-11) in which PSNEs were least common were also those on which AGG-SEM was mostly likely to time out. (We verified that AGG-SEM terminated on every game with PSNEs by checking the support size AGG-SEM was considering when it timed out. In every case, it ruled out all supports of size 1 for every agent before running out of time.)

One advantage of SEM over other Nash-equilibrium-finding algorithms is its ability to find all Nash equilibria (or all equilibria with support sizes not more than some constant). This is particularly useful when we want to understand the range of possible outcomes. For example, in [21] one of the goals was to identify the minimum and maximum revenue possible in equilibrium of position auction games. At the time, only empirical bounds on revenue were possible, because there was no algorithm available for finding all the PSNEs of an AGG. We have since tested equilibrium enumeration by searching for all PSNEs on a representative subset of these games (20 from each distribution). We found that AGG-SEM was significantly faster than NFG-SEM (see Figure 4) at enumerating the set of all equilibria. Notably, for every position auction game, AGG-SEM was able to find all PSNEs in under one CPU minute. (These games each have ten bidders, eight positions and eleven bid increments per bidder.)

5 Conclusion

We have showed that the support enumeration method can be extended to games compactly represented as AGGs. Our approach outperforms the original SEM algorithm for such games both asymptotically and in practice. Theoretically, we showed that SEM's worst-case runtime can be reduced exponentially. Our work in this vein may also be of independent interest, as it shows novel ways of exploiting AGG structure. In particular, the polynomial-time algorithm for removing dominated strategies could be useful, e.g. as

No.	Game type	Player Classes	Mean runtime (CPU s) AGG-SEM	NFG-SEM	GNM	SimpDiv
D1	Coffee shop	1	**18.00**	5032.38*	3309.73*	362.63*†
D2	Ice cream	3	131.64*	957.42*	151.59*	**0.39**
D3	Job market	1	**249.02**	6070.61*	372.96*†	1536.45*†
Position Auctions:						
D4	GFP	n	7519.90*	10878.19*	**75.73**	10750.93*
D5	Weighted GSP	n	**45.10**	96.78*	723.19*	734.56*†
Random AGGs:						
D6	Random graph	1	**68.02**	7005.34*	10580.58*	5188.02*
D7	Road graph	1	**441.11**	32103.15*	41814.79*	9507.58*
D8	Small-world graph	1	**596.75**	31750.79*	28195.09*	4665.58*
Random Graphical Games:						
D9	Random graph	n	**11953.48**	20469.50*	24337.47*	27002.81*
D10	Road graph	n	**3244.50**	32052.36*	43200.00*	43200.00*
D11	Small-world graph	n	**11356.47**	29861.96*	43200.00*	40677.67*
Overall:			**3244.28**	16520.09*	18265.93*	13176.94*

Table 1. Mean runtimes. * denotes significantly slower than fastest solver, $\alpha = 0.01$. Capped runs count as 43200s. † denotes distributions where, due to high variance, more instances (200) were necessary for statistical significance.

a preprocessing step for other equilibrium-finding algorithms. Empirically, we observed that our new algorithm was substantially (often orders of magnitude) faster, and that it almost always outperformed current state-of-the-art algorithms. Beyond this, our algorithm offers substantial advantages over existing algorithms, such as the ability to enumerate equilibria and to identify pure-strategy Nash equilibria or prove their non-existence.

We envision several extensions to AGG-SEM. One promising direction is to search for specific types of (e.g., symmetric or social-welfare-maximizing) equilibria, for example by replacing the depth-first search with branch-and-bound search. Performance could also be improved by using good heuristics to choose the order in which supports are instantiated, or even by exploring the space of supports using stochastic local search rather than tree search.

References

1. Bhat, N., Leyton-Brown, K.: Computing Nash equilibria of Action-Graph Games. In: UAI (2004)
2. Blum, B., Shelton, C., Kohler, D.: A continuation method for Nash equilibria in structured games. JAIR 25, 457–502 (2006)
3. Chernick, M.R.: Bootstrap Methods, A practitioner's guide. Wiley (1999)
4. Daskalakis, C., Schoenebeck, G., Valiant, G., Valiant, P.: On the complexity of Nash equilibria of action-graph games. In: SODA (2009)
5. Dickhaut, J., Kaplan, T.: A program for finding Nash equilibria. Mathematica J. 1, 87–93 (1991)
6. Gottlob, G., Greco, G., Scarcello, F.: Pure nash equilibria: hard and easy games. In: TARK (2003)

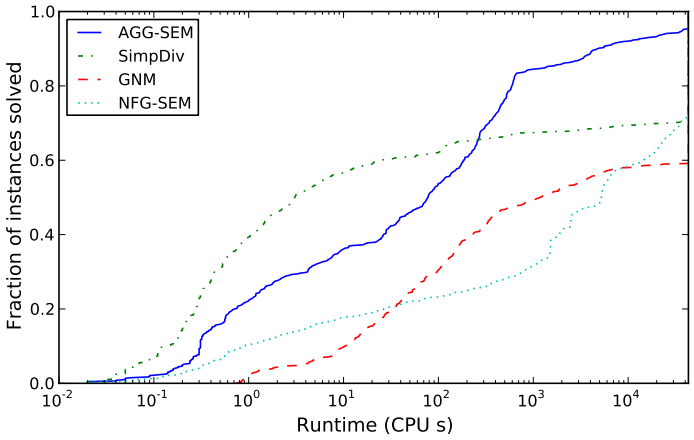

Fig. 5. Runtime CDFs for our four algorithms.

7. Govindan, S., Wilson, R.: A global Newton method to compute Nash equilibria. J. Economic Theory 110, 65–86 (2003)
8. Jiang, A.X., Leyton-Brown, K., Bhat, N.A.: Action-graph games. GEB 71, 141–173 (2011)
9. Kearns, M., Littman, M., Singh, S.: Graphical models for game theory. In: UAI (2001)
10. van der Laan, G., Talman, A.J.J., van Der Heyden, L.: Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. Mathematics of Operations Research 12, 377–397 (1987)
11. Leyton-Brown, K., Nudelman, E., Wortman, J., Shoham, Y.: Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In: AAMAS (2004)
12. Leyton-Brown, K., Tennenholtz, M.: Local-effect games. In: IJCAI (2003)
13. Lipton, R., Markakis, E.: Nash equilibria via polynomial equations. In: LATIN 2004, LNCS, vol. 2976, pp. 413–422 (2004)
14. Mangasarian, O.: Equilibrium points of bimatrix games. J. Society for Industrial and Applied Mathematics 12, 778–780 (1964)
15. McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software tools for game theory (2006), `http://econweb.tamu.edu/gambit`
16. Murtagh, B., Saunders, M.: MINOS (2010), `http://www.sbsi-sol-optimize.com`
17. Poole, D.L., Mackworth, A.K.: Artificial Intelligence. Cambridge University Press (2011)
18. Porter, R.W., Nudelman, E., Shoham, Y.: Simple search methods for finding a Nash equilibrium. GEB 63, 642–662 (2009)
19. Rosenthal, R.: A class of games possessing pure-strategy Nash equilibria. Int. J. Game Theory 2, 65–67 (1973)
20. Roughgarden, T., Papadimitriou, C.: Computing correlated equilibria in multi-player games. JACM 37, 49–56 (2008)
21. Thompson, D.R.M., Leyton-Brown, K.: Computational analysis of perfect-information position auctions. In: ACM-EC (2009)
22. Vohra, R.V.: Advanced Mathematical Economics. Routledge (2005)
23. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. JAIR 32, 565–606 (2008)