Algorithm Configuration to Investigate the Price of Anarchy

Paper ID: 11

Abstract

The Price of Anarchy (PoA) is a measure that compares the ratio between the maximum and minimum of a cost function evaluated at any Nash equilibrium of a game. There have been many results that bound the highest PoA for certain classes of games, but little has been done to explore the distribution of the PoA over a set of games. This project proposes a method to empirically evaluate this distribution using automatic algorithm configuration techniques.

1 Introduction

When self interested agents partake in a game to maximize their own utility, it is possible for them to settle into an equilibrium of some variety that is far from the social optimum. Consider the 'traveller's dilemma', where two travellers lose their luggage on a flight home. Both travellers have identical souvenirs in their bags, and the airline is willing to reimburse the travellers, but doesn't know how much the contents were worth. The airline asks the two passengers individually what the price of the trinkets were, if the passengers both report the same value, then the airline pays both of them the specified amount. If however, the reported values differ, the airline awards a small bonus to the traveller who reported the smaller value, and then gives the traveller who reported a higher value a slightly smaller amount then what was awarded to the lower value traveller. In this game, the only Nash equilibrium is where both agents report that the value of the item was as low as possible. Clearly, they both would have been happier if they both agreed to mention the highest price, since the airline would be forced to reimburse both agents the same large amount.

The Price of Anarchy (PoA) was introduced in [3] as a formal way to quantify how 'bad' a Nash equilibrium can be when compared to the optimal equilibrium with respect to a particular social function. The PoA is in the range $[1, \infty)$, with higher values indicating that if there is no coordination mechanism amongst agents to select the socially optimal equilibrium, a Nash equilibrium could be played that results in a collectively very poor outcome.

Many classes of games have tight bounds on the worst case PoA, such as congestion games and different types of auctions. However, these are all worst case measures over all possible game instances, they say nothing about what the PoA looks like on games that arise in practice. Although determining the expected PoA for a distribution could be done theoretically, it is likely to be complex, and proofs may not transfer between different distributions of game instances. As such, a method that can approximate the distribution of the PoA over several instances empirically is desired.

Since finding all Nash equilibria in a game is non-trivial and in the complexity class PPAD, it is not always feasible to compute all possible values for a game. However, there are methods that are capable of quickly finding a single Nash equilibrium in a short period of time, but there is no assurance that they will find one given a specific starting point. By applying an algorithm configurator that tunes these Nash finding algorithms towards equilibria with high and low social function valuations, the PoA can be evaluated in a feasible amount of time. Knowing what the distribution of the PoA can be over a set of possible games, designers can decide if the expected PoA of the most likely games is something they are willing to tolerate, or if they need to modify the game to produce a more favourable expected value.

2 Method

Automated algorithm configuration can be viewed as an optimization problem. The search space consists of all possible parameters that can be fed into an algorithm, and the objective function is some metric that is computed after the algorithm is executed on a set of training instances. ParamILS [2] is one such method that has been successfully used to increase the performance of SAT solvers with respect to run time, as well as improving the solution quality in scheduling problem scenarios.

As already mentioned, there exist algorithms that are capable of finding all possible equilibria in a game (or very nearly all), but they require a huge amount of time for non-trivial game sizes. Alternatively, there are other algorithms that take as input a mixed strategy profile, and are able to possibly determine some Nash equilibria of the game. The global Newton method by Govindan and Wilson [1] is one such method, and it has the advantage of being very fast.

ParamPoA is a new system that uses ParamILS to compute the PoA of an arbitrary game. It uses any Nash equilibrium solver that takes as input a mixed strategy profile. Each possible action is a parameter that can be tuned, from values between 0 and 1. Before running the solver, these parameter values are turned into a legal mixed strategy profile by normalizing the sum of the values across each player's actions. Because ParamILS is a local search method, it benefits from performing multiple independent runs, to mitigate the results of the search getting trapped in a local optimum.

The Gambit [5] package was used to compute Nash equilibria in ParamPoA. In particular the `gambit-gnm` program was used, which implements the GNM method by Govindan and Wilson, as it intakes a proposed mixed strategy profile as a starting point for computation. It should be noted that `gambit-gnm` appears to have a bug in its implementation, as it occasionally returns results that it claims are Nash equilibria, but are truly not. The cause of this error was not immediately apparent, so a small check was added to make sure that the proposed equilibrium causes the game's Lyapunov function to be equal to zero (the Lyapunov function is zero exactly when the game is in equilibrium). Whenever `gambit-gnm` returned a false equilibrium, this was not counted in ParamILS's tuning time.

3 Experiments and Results

All experiments were performed on a i7-2720QM CPU, running eight experiments in parallel. Version 2.3.5 of ParamILS was used, running on a 64-bit Linux. First, ParamPoA was applied to a number of randomly generated games to see if it was able to properly compute the PoA for these games. Following this, it was applied to a more interesting problem domain of the Generalized Second Price Auction with some success.

3.1 Random Game Verification

To verify that ParamPoA has the potential of working, GAMUT [7] was used to create 80 random two player games where each player had thirteen actions, with payoffs between -100 and 100. Using Gambit's `gambit-enummixed` method, all the extreme points of the set of equilibria were computed. ParamPoA was then run on these instances, looking for the equilibria with lowest and highest social welfare, computed as the sum of utilities for each agent.

A total of five independent runs were used for finding each min/max equilibrium, with a total of 30 seconds of tuning time per run. This time value only includes the seconds used running the algorithm - not the time taken to score individual runs when preparing an algorithm run. ParamILS will also terminate early if it thinks it is not able to improve upon its current solution. Each action parameter was quantized into the set $\{0, 0.1, 0.2, \ldots, 1.0\}$, with eleven possible choices.

The restriction to two agents was necessary, since methods for finding all possible Nash equilibria in a game take much longer when dealing with three or more agents. In each instance, the algorithm configuration method was able to find an equilibrium of each the lowest and highest social welfare, computing the correct value for the PoA. However, using the enumeration method in Gambit required significantly more computing time; on average ParamPoA performed better than the best case for the enumeration method. Table 1 provides timing information for these experiments.

3.2 The Generalized Second Price Auction

The Generalized Second Price Auction (GSP), commonly known as the AdWords auction [6], is receiving a great deal of attention from researchers, due

Table 1: 13x13 Random Games - Runtime for Computation of PoA (seconds)

	Min	Max	Mean	Std. Dev.
`gambit-enummixed`	219	86,448	12,111	15,691
ParamPoA	103	462	213	68

to it's heavy use in e-commerce and other interesting properties. It is not dominant strategy truthful like the famous Vickrey-Clark-Groves mechanism, nor does it maximize social welfare. In the GSP, each player has a valuation v_i, and they place a single bid b_i for positions in a queue. The auctioneer awards the n^{th} most valuable position to the player who placed the n^{th} highest bid, but charges them the $n+1^{th}$ highest bid. In the AdWords setting, these positions correspond to rankings in the list of sponsored results returned from a search query. It's natural to assume that the first position will get the highest number of clicks from users, so it is the most valuable, as it has the potential to draw the most visitors to a bidder's website.

Leme and Tardos [4] proved bounds on the PoA for the GSP, under Pure, Mixed and Bayes-Nash equilibria. They restrict themselves to agents that are conservative (since bidding above your valuation v_i is a weakly-dominated strategy), and define the quality of an equilibrium as $\sum \alpha_i v_i$, where α_i is the relative quality of the position that was awarded to agent i. Using this measure, they come up with a bound of 4 for mixed-Nash, and 8 for Bayes-Nash equilibria.

The GSP can be formulated as a normal form game, where bids a player make correspond to the actions that they can take. This moves the game from a continuous game to a discrete one, since in practice bidders can't make bids with increments smaller than a penny, this is an acceptable trade off. Each player's utility is defined as $u_i = \alpha_i(v_i - b_i)$. The social welfare of an outcome in this game is $\sum u_i$. If restricted to agents that don't play weakly dominated strategies, further imposition of the rule that in the case that $b_i > v_i$, $u_i = -\infty$ is required. In GSP, when the social welfare of an outcome is high, it means that agents were able to gain a large benefit by gaining a high value position while paying less to the auctioneer.

3.2.1 2 Player GSP

160 instances of two player GSP (2-GSP) were created, with the valuations for bidders drawn uniformly from integers 1 to 10, with an action space of integer bids between 0 to 10. Each position's value was uniformly drawn from the range of integers 0 to 10. ParamPoA ran a total of five independent runs for finding each min/max equilibrium on each instance, with a total of 60 seconds of tuning time for each run. In all but six of the generated instances, ParamPoA was able to compute the correct value, verified by solving the game using `gambit-enummixed`. Because the action space is quite small, these instances can be solved rapidly using this method, so timing information is not that interesting. When ParamPoA failed, it always underestimated the correct

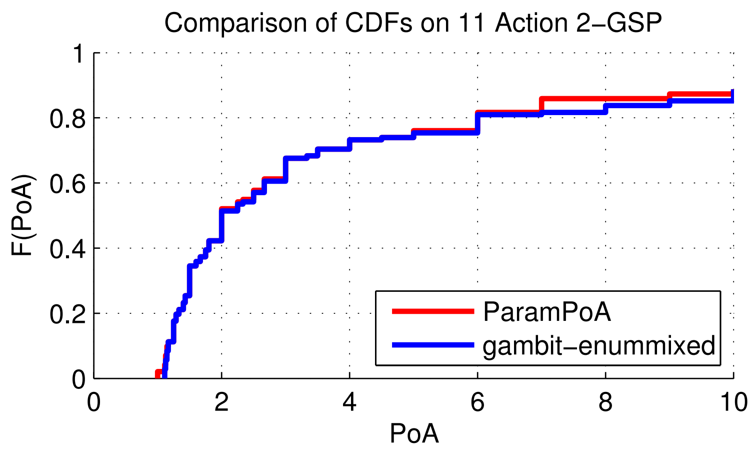

Figure 1: CDF of the PoA computed using ParamPoA and `gambit-enummixed` on 160 random GSP instances with 2 players, 11 actions each.

value. Figure 1 shows the CDF of the PoA obtained when using ParamPoA as compared to the actual CDF for the generated instances.

Next, 160 instances of 2-GSP were created again, but this time the action space was nearly doubled. Players' valuations took on random values in increments of 0.5 from 0.5 to 10, and could place a bid in 0.5 increments from 0 to 10. The valuations for each position were also chosen from this interval from 0 to 10. Given the larger action space, ParamPoA was tuned for 120 seconds, still performing five independent runs looking for min/max values per instance. In these experiments, ParamPoA was able to compute the correct PoA in all but 24 instances. Since these games often have a large number of dominated strategies, in nearly all the instances the computation of `gambit-enummixed` takes just a few seconds, but rarely it can take a number of hours. Table 2 shows the differences in times for these calculations.

Just like in the previous case of eleven action 2-GSP, when ParamPoA failed it always computed a PoA that was lower then the correct value. Figure 2 shows a comparison of the true empirical distribution against the one computed by ParamPoA.

Table 2: 2-GSP with 21 actions - Runtime for Computation of PoA (seconds)

	Min	Max	Mean	Std. Dev.
`gambit-enummixed`	0.007	42,482	1,203	5,754
ParamPoA	538	984	829	87

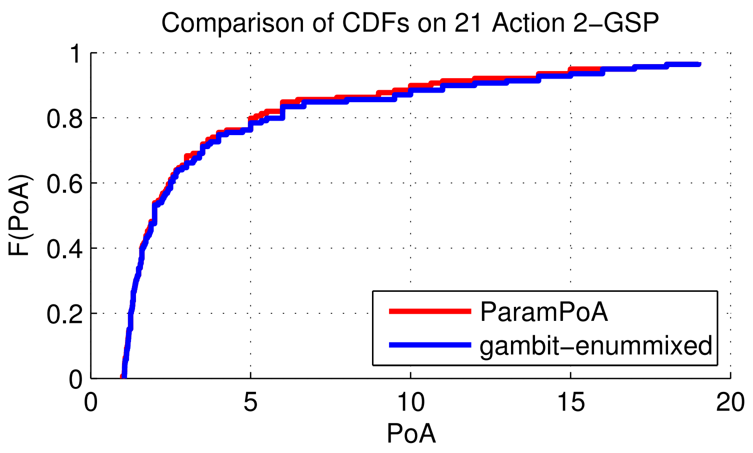

Figure 2: CDF of the PoA computed using ParamPoA and `gambit-enummixed` on 160 random GSP instances with 2 players, 21 actions each.

3.2.2 3 or more Player GSP

Given the success with two player games, ParamPoA was tested on three player GSP games. Similar to the first instances of 2-GSP, players had eleven actions, with valuations between 1 and 10 for both the bidders and the positions. ParamPoA again used five independent runs, but used a larger tuning time of 120 seconds since the equilibrium calculation would now take slightly longer. On the 160 instances, ParamPoA failed to achieve significant results. It almost always computed the PoA to be 1, but this was verified to be wrong by running `gambit-enumpoly` (which is not guaranteed to find all equilibria in a game) for a short while, and finding a number of equilibria with different social welfares. Even when the tuning time was extended up to 600 seconds, there was no discernible difference in the resulting computation.

4 Future Work

Currently, this method is only used to sample the PoA from a specified distribution of games. Since finding an upper bound on the PoA is of interest, it should be relatively straightforward to extend this system for finding an instance that has a large PoA by adding another layer of algorithm configuration. Instead of generating the problem instances randomly, the inputs to the game generator could be paramaterized, and an algorithm tuner could then find the settings that would generate a game with the highest PoA.

The `gambit-gnm` implementation (or the implementation of ParamPoA) needs to be examined carefully to search for the reason behind the method returning false Nash equilibria. In addition, different Nash finding algorithms need to be evaluated for their ability to find low and high valued equilibria, as well as the effect of different algorithm configurators.

Even though ParamPoA failed to work on GSP instances with more than two players, a few brief experiments demonstrated that it manages to achieve some success with three or more player random games. Note again that not all equilibria were found for these games, but the values produced by ParamPoA were at least as good as the equilibria returned from `gambit-enumpoly`. More effort needs to be placed into the investigation of when ParamPoA fails to see if this is an isolated incident, or if it will not work on special classes of games.

5 Conclusion

Algorithm configuration was successfully used to tune a Nash equilibrium solver for some games, and was able to compute the distribution of the PoA to an acceptable level of accuracy for these games. However, for some larger games, the configuration process failed to achieve any meaningful results. Despite this, ParamPoA is still useful, as it is able to provide a very close approximation of the PoA of two player games faster then solving the game completely. Further investigations need to be made to see if ParamPoA can be extended to remove its limitations, allowing for PoA calculations of arbitrary game instances in a practical amount of time.

References

[1] S. Govindan and R. Wilson. A global newton method to compute nash equilibria. *Journal of Economic Theory*, 110(1):65–86, 2003.

[2] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[3] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on Theoretical aspects of computer science*, pages 404–413. Springer-Verlag, 1999.

[4] R.P. Leme and E. Tardos. Pure and bayes-nash price of anarchy for generalized second price auction. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 735–744. IEEE, 2010.

[5] R. D. Mckelvey, A. M. Mclennan, and T. L. Turocy. Gambit: Software Tools for Game Theory, 2010. `http://www.gambit-project.org`.

[6] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 264–273. IEEE, 2005.

[7] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 880–887. IEEE Computer Society, 2004.