Efficient Decomposition of Bimatrix Games (Extended Abstract)*

Xiang Jiang & Arno Pauly
Computer Laboratory
University of Cambridge, United Kingdom
Arno.Pauly@cl.cam.ac.uk

Exploiting the algebraic structure of the set of bimatrix games, a divide-and-conquer algorithm for finding Nash equilibria is proposed. The algorithm is fixed-parameter tractable with the size of the largest irreducible component of a game as parameter. An implementation of the algorithm is shown to yield a significant performance increase on inputs with small parameters.

1 Introduction

A bimatrix game is given by two matrices (A,B) of identical dimensions. The first player picks a row i, the second player independently picks a column j. As a consequence, the first player receives the payoff A_{ij}, the second player B_{ij}. Both player are allowed to randomize over their choices, and will strive to maximize their expected payoff. A Nash equilibrium is a pair of strategies, such that no player can improve her expected payoff by deviating unilaterally.

If the payoff matrices are given by natural numbers, then there always is a Nash equilibrium using only rational probabilities. The computational task to find a Nash equilibrium of a bimatrix game is complete for the complexity class PPAD [21, 7, 6]. PPAD is contained in FNP, and commonly believed to exceed FP. In particular, it is deemed unlikely that a polynomial-time algorithm for finding Nash equilibria exists.

The next-best algorithmic result to hope for could be a fixed-parameter tractable (fpt) algorithm [8], that is an algorithm running in time $f(k)p(n)$ where n is the size of the game, p a polynomial and k a parameter. For such an algorithm to be useful, the assumption the parameter were usually small needs to be sustainable. The existence of fpt algorithms for finding Nash equilibria with various choices of parameters has been studied in [9, 13, 10].

In the present paper we demonstrate how *products* and *sums* of games – and their inverse operations – can be used to obtain a divide-and-conquer algorithm to find Nash equilibria. This algorithm is fpt, if the size of the largest component not further dividable is chosen as a parameter. *Products* of games were introduced in [23] as a means to classify the Weihrauch-degree [4, 3, 5, 14] of finding Nash equilibria for real-valued payoff matrices. *Sums* appear originally in the PhD thesis [24] of the second author; the algorithm we discuss was implemented in the Bachelor's thesis [15] of the first author.

2 Products and Sums of Games

Both products and sums admit an intuitive explanation: The product of two games corresponds to playing both games at the same time, while the sum involves playing *matching pennies* to determine which game to play, with one player being rewarded and the other one punished in the case of a failure to agree.

*A full version including proofs omitted here is available as [16].

F. Mogavero, A. Murano, and M.Y. Vardi (Eds.):
2nd Workshop on Strategic Reasoning 2014 (SR14)
EPTCS 146, 2014, pp. 75–81, doi:10.4204/EPTCS.146.10

2.1 Products

In our definition of products, we let $[\ ,\ ]:\{1,\ldots,n\}\times\{1,\ldots,m\}\to\{1,\ldots,nm\}$ denote the usual bijection $[i,j]=(i-1)n+j$. The relevant values of n,m will be clear from the context. We point out that $[\ ,\ ]$ is polynomial-time computable and polynomial-time invertible.

Definition 1. Given an $n_1\times m_1$ bimatrix game (A^1,B^1) and an $n_2\times m_2$ bimatrix game (A^2,B^2), we define the $(n_1n_2)\times(m_1m_2)$ product game $(A^1,B^1)\times(A^2,B^2)$ as (A,B) with $A_{[i_1,i_2][j_1,j_2]}=A^1_{i_1j_1}+A^2_{i_2j_2}$ and $B_{[i_1,i_2][j_1,j_2]}=B^1_{i_1j_1}+B^2_{i_2j_2}$.

Theorem 2. If (x^k,y^k) is a Nash equilibrium of (A^k,B^k) for both $k\in\{1,2\}$, then (x,y) is a Nash equilibrium of (A,B), where $x_{[i_1i_2]}=x^1_{i_1}x^2_{i_2}$ and $y_{[m_1m_2]}=y^1_{m_1}y^2_{m_2}$.

Theorem 3. If (x,y) is a Nash equilibrium of (A,B), then (x^1,y^1) given by $x^1_i=\sum_{l=1}^{n_2}x_{[i,l]}$ and $y^1_j=\sum_{l=1}^{m_2}y_{[j,l]}$ is a Nash equilibrium of (A^1,B^1).

2.2 Sums

The sum of games involves another parameter besides the two component games, which just is a number exceeding the absolute value of all the payoffs.

Definition 4. Given an $n_1\times m_1$ bimatrix game (A^1,B^1) and an $n_2\times m_2$ bimatrix game (A^2,B^2), we define the $(n_1+n_2)\times(m_1+m_2)$ sum game $(A^1,B^1)+(A^2,B^2)$ via the constant $K>\max_{i,j}\{|A_{i,j}|,B_{i,j}|\}$ as (A,B) with:

$$A_{i,j}=\begin{cases}A^1_{ij} & \text{if } i\le n_1, j\le m_1\\ A^2_{(i-n_1),(j-m_1)} & \text{if } i>n_1, j>m_1\\ K & \text{otherwise}\end{cases}$$

$$B_{i,j}=\begin{cases}B^1_{ij} & \text{if } i\le n_1, j\le m_1\\ B^2_{(i-n_1),(j-m_1)} & \text{if } i>n_1, j>m_1\\ -K & \text{otherwise}\end{cases}$$

Lemma 5. Let (x,y) be a Nash equilibrium of $(A^1,B^1)+(A^2,B^2)$. Then $0<\left(\sum_{i=1}^{n_1}x_i\right)<1$ and $0<\left(\sum_{j=1}^{m_1}y_j\right)<1$.

Theorem 6. If (x,y) is a Nash equilibrium of $(A^1,B^1)+(A^2,B^2)$, then a Nash equilibrium (x^1,y^1) of (A^1,B^1) can be obtained as $x^1_i=\frac{x_i}{\sum_{l=1}^{n_1}x_l}$ and $y^1_j=\frac{y_i}{\sum_{l=1}^{m_1}y_l}$.

Theorem 7. Let (x^k,y^k) be a Nash equilibrium of (A^k,B^k) resulting in payoffs (P^k,Q^k) for both $k\in\{1,2\}$. Then (x,y) is a Nash equilibrium of $(A^1,B^1)+(A^2,B^2)$, where $x_i=x^1_i\frac{K-Q^2}{2K-Q^1-Q^2}$ for $i\le n_1$, $x_i=x^2_{i-n_1}\frac{K-Q^1}{2K-Q^1-Q^2}$ for $i>n_1$, $y_j=y^1_j\frac{K-P^2}{2K-P^1-P^2}$ for $j\le m_1$, $y_j=y^2_{j-m_1}\frac{K-P^1}{2K-P^1-P^2}$ for $j>m_1$.

If a game is iteratively decomposed into sums, the resulting structure corresponds to a Blackwell game [1, 19] of finite length. The reasoning underlying the theorems above then provides a means of backwards-induction to show that such games always admit Nash equilibria without a direct appeal to their normal form version. The latter observation is the foundation for [18, Corollary 8].

3 Examples

In order to illuminate both how the operations work, and how the component games can be recovered from the compound game, we shall briefly consider some examples. As all relevant features already appear for zero-sum games, we shall restrict the examples to this case, and suppress explicit reference to the second player's payoffs.

$$A := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix} \qquad B := \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

$$A \times B = \begin{pmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 3 & 4 & 5 & 6 & 4 & 5 & 6 & 7 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 2 & 0 & 1 & 0 & 1 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 & 2 & 3 & 2 & 3 & 3 & 4 & 3 & 4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 5 & 5 & 5 & 5 \\ 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 1 & 2 & 3 \\ 5 & 2 & 3 & 4 & 4 & 1 & 2 & 3 & 5 & 2 & 3 & 4 \\ 5 & 2 & 3 & 4 & 6 & 3 & 4 & 5 & 7 & 4 & 5 & 6 \end{pmatrix} \qquad \begin{aligned} A_{ij} + B_{11} &= (A \times B)_{3i-2,j} \\ B_{ij} + A_{21} &= (A \times B)_{3+i,4j-3} \end{aligned}$$

As demonstrated by the colour-markings in the product game, if a game is indeed a product game, then the payoffs of the components can be read off from the payoff matrix of the composed game (plus some constant). Indeed, there are many different positions where the component games are found. Verifying that a game indeed is a product requires to ensure the consistency of these answers.

$$A + B = \begin{pmatrix} 1 & 2 & 3 & 4 & K & K & K \\ 0 & 1 & 0 & 1 & K & K & K \\ 2 & 2 & 2 & 2 & K & K & K \\ 4 & 1 & 2 & 3 & K & K & K \\ K & K & K & K & 0 & 0 & 0 \\ K & K & K & K & 1 & 0 & 1 \\ K & K & K & K & 1 & 2 & 3 \end{pmatrix}$$

Recovering the component games from a sum is even simpler: The payoff matrices are found in the left-upper and right-lower corner, while the remaining two cornes are covered by a suitable constant K. The latter allows us to determine the precise size of the corners.

4 The algorithm

Our basic algorithm proceeds as follows: To solve a game (A,B)

1. test whether (A,B) is the sum of (A^1,B^1) and (A^2,B^2) via some constant K. If yes, solve (A^1,B^1) and (A^2,B^2) and combine their Nash equilibria to an equilibrium of (A,B) via Theorem 7. If no,

2. test whether (A,B) is the product of (A^1,B^1) and (A^2,B^2). If yes, solve (A^1,B^1) and (A^2,B^2) and combine their Nash equilibria to an equilibrium of (A,B) via Theorem 2. If no,

3. find a Nash equilibrium of (A,B) by some other means.

For some $n \times m$ game (A,B) let let $S(A,B)$ denote its size, i.e. $S(A,B) = nm$, and let $\lambda(A,B)$ be the size of the largest game for which 3. in our algorithm is called. Let $f(k)$ be the time needed for the external algorithm called in 3. on a game of size k. Then the runtime of our algorithm is bounded by $O(S^2 f(\lambda))$, in particular, it is an fpt-algorithm:

Testing whether a game is a sum, and computing the components, if applicable, can be done in linear time: As the value K has to appear in the corner, one look up whether two suitable rectangular regions have payoffs K and $-K$. If this is the case, the remaining two rectangular regions contain the two subgames, provided they do not contain payoffs p with $|p| \geq K$. The sum of the sizes of the components is less than the size of the original game. Finally, combining Nash equilibria can be done in linear time, too. Only this case for yield quadratic runtime.

Whether a game is a product of factors of a fixed size can also be tested in linear time. Essentially, the payoffs of the putative component games are found as differences between corresponding payoffs in the original game. Then the product of the two component games can be computed, and finally compared to the original game to verify the decomposition. Testing the different possible factors adds an additional factor $\sqrt{S}$ for this part. The product of the sizes of the factors is equal to size of the original game. Again, combining the Nash equilibria takes linear time. The underlying recurrence relation on its own would yield a time bound of $O(S^{1.5} \log S)$, hence the quadratic runtime bound from the first relation is dominating.

Note that a game cannot simultaneously be the result of a sum and a product of smaller games. Thus, a full decomposition (and the size of the largest component) of a game is independent of the order in which decomposability is tested.

As a slight modification of our algorithm, one can eliminate (iteratively) strictly dominated strategies at each stage of the algorithm. We recall that a strategy i of some player is called strictly dominated by some other strategy j, if against any strategy chosen by the opponent, i provides its player with a strictly better payoff than j. A strictly dominated strategy can never be used in a Nash equilibrium. It is easy to verify that a game decomposable as a sum never has any strictly dominated strategies, but may occur as the result of the elimination of such strategies. Hence, including an elimination step for each stage increases the potential for decomposability.

Proposition 8. Elimination of strictly dominated strategies commutes with decomposition of products, i.e. the reduced form of the product is the product of the reduced forms of the factors.

The algorithm remains fpt if such a step is included, using e.g. the algorithm presented in [17]. The exponent would presumably increase to $O(S^4 f(\lambda))$ though. Discussion of complexity issues regarding removal of dominated strategies can be found in e.g. [2, 22].

5 Empirical evaluation

Only a small fraction of the bimatrix games of a given size and bounded integer payoffs will be decomposable by our techniques, this limiting the applicability of the algorithm in Section 4. In particular, if sample games were drawn from a uniform distribution, one should not expect any speedup using decomposability-tests. However, to some extent we can expect patterns in the definitions of real-world

game situation to increase the decomposability of the derived bimatrix games. For example, the structure of Poker-style games implies decomposability, as can be concluded from the considerations[1] in [11]. Note that an explicit understanding of such patterns is not required to benefit from our algorithm – a reasonable expectation that suitable patterns could occur is enough to justify the use of our algorithm, which then identifies the actual patterns.

To obtain a first impression whether using the decomposition algorithm is indeed beneficial for computing Nash equilibria, a collection of 100 random decomposable games was created. Each game has 95-105 strategies per player, and payoff values range from 0 to 50. The decomposability was ensure by creating a random tree representing the relevant decomposition structure first, using probabilities of 0.4 each for sum and product decomposition, and of 0.2 for an elimination of strictly dominated strategies step. The height of the trees was limited to 80, additionally vertices corresponding to games of size up to 6 were turned into leaves. At the leaves, the payoffs were chosen uniformly subject to the constraints derived from the structure and the overall constraint of payoff values being between 0 and 50. Finally, the corresponding bimatrix games were computed. As the payoffs for both players were chosen independently, the expected fraction of zero-sum games in the sample set is negligible.

Both as a benchmark, and in order to compute Nash equilibria of the irreducible component games, the tool GAMBIT [20] was used. GAMBIT offers a variety of algorithm for computing Nash equilibria of bimatrix games, we used:

1. gambit-enummixed: using extreme point enumeration
2. gambit-gnm: using a global Newton method approach
3. gambit-lcp: using linear complementarity
4. gambit-simpdiv: using simplicial subdivision

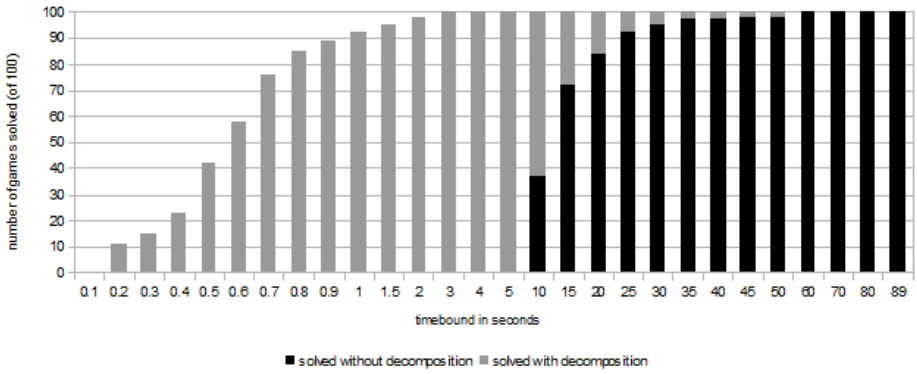

Figure 1: gambit-gnm

Figures 1. & 2. show for gambit-gmn and gambit-lcp how many of our decomposable example games could be solved in some given time bound (per game, not total) using only the GAMBIT algorithm directly, or exploiting decomposition implemented in C++ first. Despite the fact that our decomposition algorithm was not optimized, it turned out that using decomposition almost all games could be solved

[1] Making decisions on whether to *fold*, *call* or *raise* corresponds to choosing the type of the remaining game, i.e. a sum decomposition. Similarly, the cards chosen by chance induce a product decomposition of the expected values of the game.

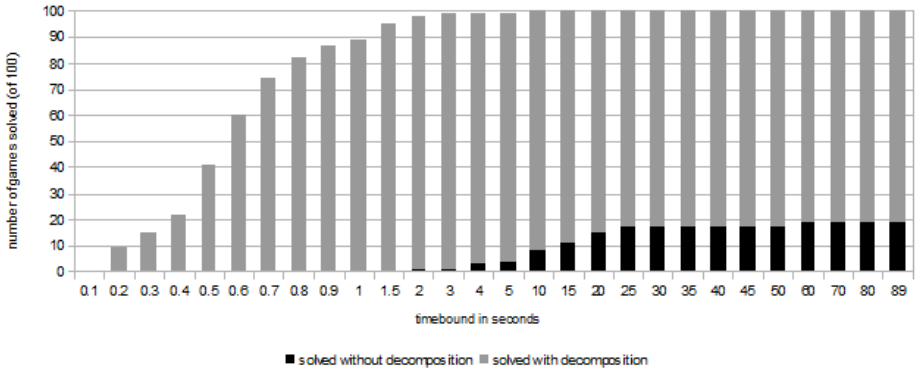

Figure 2: gambit-lcp

in under 3 seconds, whereas even gambit-gnm[2] as the fastest GAMBIT algorithm on the sample took 30 seconds for a similar feat. Neither gambit-enummixed nor gambit-subdiv were able to solve any of the example games in less than 90 seconds without decomposition. Note that the GAMBIT algorithms generally include elimination of strictly dominated strategies as well, so this alone cannot account for the differences. Thus, there is clear indication that on suitable data, exploiting the algebraic structure underlying the decomposition algorithm yields a significant increase in performance.

References

[1] D. Blackwell (1969): *Infinite G_δ games with imperfect information*. Zastowania Matematyki Applicationes Mathematicae.

[2] Felix Brandt, Markus Brill, Felix Fischer & Paul Harrenstein (2011): *On the Complexity of Iterated Weak Dominance in Constant-sum Games*. Theory of Computing Systems 49(1), pp. 162–181. Available at `http://dx.doi.org/10.1007/s00224-010-9282-7`.

[3] Vasco Brattka, Matthew de Brecht & Arno Pauly (2012): *Closed Choice and a Uniform Low Basis Theorem*. Annals of Pure and Applied Logic 163(8), pp. 968–1008, doi:10.1016/j.apal.2011.12.020.

[4] Vasco Brattka & Guido Gherardi (2011): *Effective Choice and Boundedness Principles in Computable Analysis*. Bulletin of Symbolic Logic 1, pp. 73 – 117, doi:10.2178/bsl/1294186663. ArXiv:0905.4685.

[5] Vasco Brattka, Stéphane Le Roux & Arno Pauly (2012): *On the Computational Content of the Brouwer Fixed Point Theorem*. In S.Barry Cooper, Anuj Dawar & Benedikt Löwe, editors: How the World Computes, Lecture Notes in Computer Science 7318, Springer Berlin Heidelberg, pp. 56–67, doi:10.1007/978-3-642-30870-3_7.

[6] Xi Chen, Xiaotie Deng & Shang-Hua Teng (2009): *Settling the Complexity of Computing Two-player Nash Equilibria*. J. ACM 56(3), pp. 14:1–14:57, doi:10.1145/1516512.1516516.

[7] Constantinos Daskalakis, Paul Goldberg & Christos Papadimitriou (2006): *The Complexity of Computing a Nash Equilibrium*. In: 38th ACM Symposium on Theory of Computing, pp. 71–78, doi:10.1145/1132516.1132527.

[8] Rod Downey & Michael Fellows (1999): *Parameterized Complexity*. Springer, doi:10.1007/978-1-4612-0515-9.

[2] Which is based on [12].

[9] Vladimir Estivill-Castro & Mahdi Parsa (2009): *Computing Nash equilibria Gets Harder – New Results Show Hardness Even for Parameterized Complexity*. In Rod Downey & Prabhu Manyem, editors: *CATS 2009*, *CRPIT* 94.

[10] Vladimir Estivill-Castro & Mahdi Parsa (2011): *Single Parameter FPT-Algorithms for Non-trivial Games*. In Costas Iliopoulos & William Smyth, editors: *Combinatorial Algorithms*, *Lecture Notes in Computer Science* 6460, Springer, pp. 121–124. Available at `http://dx.doi.org/10.1007/978-3-642-19222-7_13`.

[11] Andrew Gilpin, Javier Pena, Samid Hoda & Tuomas Sandholm (2007): *Gradient-based algorithms for finding Nash equilibria in extensive form games*. In: *Proceedings of the 18th Int Conf on Game Theory*, doi:10.1007/978-3-540-77105-0_9.

[12] Srihari Govindan & Robert Wilson (2003): *A Global Newton Method to Compute Nash Equilibria*. *Journal of Economic Theory* 110(1), pp. 65–86, doi:10.1016/S0022-0531(03)00005-X.

[13] Danny Hermelin, Chien-Chung Huang, Stefan Kratsch & Magnus Wahlström (2010): *Parameterized Two-Player Nash Equilibrium*. *CoRR* abs/1006.2063. Available at `http://arxiv.org/abs/1006.2063`.

[14] Kojiro Higuchi & Arno Pauly (2013): *The degree-structure of Weihrauch-reducibility*. *Logical Methods in Computer Science* 9(2), doi:10.2168/LMCS-9(2:2)2013.

[15] Xiang Jiang (2011): *Efficient Decomposition of Games*. Bachelor's thesis, University of Cambridge.

[16] Xiang Jiang & Arno Pauly (2012): *Efficient Decomposition of Bimatrix Games*. http://arxiv.org/abs/1212.6355.

[17] Donald Knuth, Christos Papadimitriou & John Tsitsiklis (1988): *A note on strategy elimination in bimatrix games*. *Operations Research Letters* 7(3), pp. 103–107, doi:10.1016/0167-6377(88)90075-2.

[18] Stéphane Le Roux & Arno Pauly (2014): *Infinite sequential games with real-valued payoffs*. arXiv:1401.3325.

[19] Donald A. Martin (1998): *The Determinacy of Blackwell Games*. *Journal of Symbolic Logic* 63(4), pp. 1565–1581, doi:10.2307/2586667.

[20] Richard McKelvey, Andrew McLennan & Theodore Turocy (2010): *Gambit: Software Tools for Game Theory*. http://www.gambit-project.org. Version 0.2010.09.01.

[21] Christos H. Papadimitriou (1994): *On the complexity of the parity argument and other inefficient proofs of existence*. *Journal of Computer and Systems Science* 48(3), pp. 498–532, doi:10.1016/S0022-0000(05)80063-7.

[22] Arno Pauly (2009): *The Complexity of Iterated Strategy Elimination*. arXiv:0910.5107.

[23] Arno Pauly (2010): *How Incomputable is Finding Nash Equilibria?* *Journal of Universal Computer Science* 16(18), pp. 2686–2710, doi:10.3217/jucs-016-18-2686.

[24] Arno Pauly (2012): *Computable Metamathematics and its Application to Game Theory*. Ph.D. thesis, University of Cambridge.

Acknowledgements

We are grateful for various helpful comments from referees. One comment in particular was instrumental in improving the runtime analysis of our algorithm to a quadratic exponent.