Experimental Studies in General Game Playing: An Experience Report

Jakub Kowalski, Marek Szykuła
University of Wrocław, Poland
jko@cs.uni.wroc.pl, msz@cs.uni.wroc.pl

Abstract

We describe nearly fifteen years of General Game Playing experimental research history in the context of reproducibility and fairness of comparisons between various GGP agents and systems designed to play games described by different formalisms. We think our survey may provide an interesting perspective of how chaotic methods were allowed when nothing better was possible. Finally, from our experience-based view, we would like to propose a few recommendations of how such specific heterogeneous branch of research should be handled appropriately in the future. The goal of this note is to point out common difficulties and problems in the experimental research in the area. We hope that our recommendations will help in avoiding them in future works and allow more fair and reproducible comparisons.

Introduction

As an alternative for the research trying to solve particular human-created games, like chess (Campbell, Hoane, and Hsu 2002), checkers (Schaeffer et al. 2007), or go (Silver et al. 2016), General Game Playing (GGP) domain has been established to learn computers to play any given game, in particular, the one with previously unknown rules. Although the main idea can be traced back to the famous General Problem Solver from 1959 (Newell, Shaw, and Simon 1959), the area really started developing with the call for the Stanford's GGP competition in 2005 (Genesereth, Love, and Pell 2005).

Since that time, more GGP systems and formal game description languages emerged, and for all of those systems, many agent's descriptions, algorithms, and theoretical analyses have been published. However, the domain fragmentation and complexity of used solutions cause the recurring issue of the once-done-never-rerun experiments. Thus, they were reported, and in most cases, could not be reproduced or even verified in any convincing way.

In this paper, we are presenting an excerpt from experimental studies from nearly fifteen years of general game playing history in the context of research reproducibility (Gundersen, Gil, and Aha 2018). We are investigating the fairness of comparisons between various GGP agents and also systems designed to play games described by different languages, providing a large number of descriptive examples. In particular, we demonstrate a very recent collection of works that visualizes how important it is to ensure that the results are well described and verifiable – which in the other case may even cause a debate levering the correctness of the paper.

From a broad perspective, our goal is to provide a perspective of chaotic methods that were allowed when nothing better was possible. Finally, from our experience-based view, we would like to propose a few recommendations of how such specific heterogeneous branch of research should be handled appropriately in the future, and how to avoid at least some of the issues indicated in this report.

General Game Playing Domain

The oldest General Game Playing (GGP) approaches focused on generalizations of chess-like games, which can be consistently described under one formalism, and then played or even procedurally generated (Pitrat 1968; Pell 1992).

The real advancement, and formal establishing of GGP as a proper research domain, was due to the announcement of Stanford's Game Description Language (GDL) and associated International General Game Playing Competition (IGGPC) in 2005 (Genesereth, Love, and Pell 2005; Love et al. 2006).

GDL can describe any turn-based, finite, and deterministic n-player game with perfect information. It is a high-level, strictly declarative logic language, based on Datalog (Abiteboul, Hull, and Vianu 1995). The language does not provide any predefined functions, so every predicate encoding the game structure like a board or a card deck, or even arithmetic operators, must be defined explicitly from scratch. Of course, this should not be seen as a drawback of GDL, because due to that, it describes games in a very knowledge-free way, stimulating the development of knowledge-inference methods from a general game description.

For quite a long time, the entire GGP domain was equated to Stanford's GGP (Genesereth and Thielscher 2014). This is definitely the most influential system, resulting in many valuable research and algorithm advancements (e.g., for MCTS (Browne et al. 2012)). Multiple GDL extensions have

been designed, e.g., GDL-II introducing randomness and imperfect information (Thielscher 2010), rtGDL removing from the system the turn-based restriction (Kowalski and Kisielewicz 2016), or GDL-III for describing games with imperfect information and introspection (Thielscher 2017).

After Stanford's GGP gained popularity, more general game playing formalisms had been designed. Some of them are aiming to be as general as GDL but faster or more concise. Some are describing significantly simpler classes of games, but because of that, giving more information to the agents. Finally, some are entirely unrelated, aiming to represent, e.g., real-time video games.

A variety of GGP languages

In this section, we will briefly introduce other GGP formalisms that will be mentioned throughout the paper.

TOSS (Kaiser and Stafiniak 2011a), proposed as a GDL alternative, is based on first-order logic with counting. The language structure allows more accessible analysis of games, as it is possible to generate heuristics from existential goal formulas.

Simplified Boardgames (Björnsson 2012) describes chess-like games using regular expressions to encode movement of pieces. Its expressiveness is very limited, but the language is concise, and conceptually well defined. Thus it is relatively simple to use in some advanced tasks like a procedural generation of game rules (Kowalski and Szykuła 2016).

Ludi system was designed for the sake of the procedural generation of games from a restricted domain of combinatorial games (Browne and Maire 2010). Its successor Ludii (Piette et al. 2019a) (currently under development), aims to describe any traditional strategy game throughout recorded human history. The system is planned to be used to chart the historical development of games and also explore their role in the development of human culture.

Regular Boardgames (RBG) (Kowalski et al. 2019b) is a novel GGP system based on the theory of regular languages. Like GDL, it provides only a few generic mechanisms that already allows describing the class of all finite deterministic turn-based games with perfect information. But instead of logic, it uses a transition system of modifying the game state that naturally occurs in games. Therefore, it allows concise encoding and effective playing games with complex rules and with large branching factor such as arimaa, go, and international checkers – which is impossible in, e.g., Stanford's GDL.

On the contrast of the beforementioned formalisms, General Video Game AI (GVGAI) domain is strictly focused on representing and playing real-time Atari-like video games (Perez-Liebana et al. 2019a; 2019b). It recently gained much popularity due to its (relative) simplicity, so the organizers run a few multitrack competitions per year. Instead of game descriptions, as in the case of Stanford's GGP, the GVGAI competition framework provides a forward model as a programming structure. Thus, the agents do not need to (and cannot) parse and understand game semantic directly from its rules.

Comparing Agents Efficiency in Stanford's GGP

To visualize the Stanford's GGP domain difficulties and progress in the quality of presenting results, we discuss the development of the branch devoted to improving the reasoning efficiency of GDL. In contrast with the theoretical research, which usually does not require experiments or they are performed to visualize the effect of the proposed solution unrelated to other solutions, this branch should always provide reliable comparisons between existing approaches.

The growing number of competition-ready GGP agents and problems with slow standard Prolog-based GDL reasoning resulted in many proposals to speed-up the logic resolution and compute game states faster. Soon, this became a crucial aspect of both research progress and success during the tournaments. Because GDL is a subset of first-order-logic Datalog language with specific keywords and information flow added, the possibilities are vast; just to mention compilations to other languages, partial computation of possible fact occurrences (*instantiation*), optimizing data structures for fast querying, or even putting GDL reasoners into a hardware (Siwek et al. 2018).

Context and Limitations

For the sake of completeness, we would like to point out reasons that shape Stanford's GDL research as it was.

Probably, the most important issue was full freedom in available technologies. The specified http-based communication protocol allows using any programming language (or a mix of programming languages), any third-party resources, and any hardware – which caused situations like a game between agents running on a single laptop vs. on a cluster of computers.

The requirement that the game manager communicates with the player via the TCP/IP connection causes another set of issues. For example, all players had to take into account the communication lag. Thus, all messages were usually sent a few seconds before the timelimit. Given that for some games these limits oscillates in 15-30 seconds, these few seconds may have some non-negligible impact on the player's performance, yielding a disadvantage depending on the geographical localization. Also, the communication architecture forces player agents to be servers (while the game manager was a client-type application), which requires public IP to play against other online agents.

Lastly, research on Stanford's GGP started in 2005 and grew rapidly until about 2015–2017. During this time, the necessity for fully reproducible research, although needed, was not as clearly defined as today. Also, managing ongoing open-source projects was slightly more complicated. The most widely used open-source GGP Base project containing i.a. example players, games repository, rules validator, game manager allowing agent vs. agent and agent vs. human games, was started in late 2013 (Schreiber 2013).

Overview of the Research

Let us then briefly present and analyze several works concerning the efficiency improvements from the point of view

of experiments and their supposed reproducibility.

• (Waugh 2009) is the first published GDL compiler. The author translates the GDL descriptions into a compilable C++ code. Experiments compare the efficiency of the proposed solution against example YAP Prolog-based GDL interpreter on four games: tic-tac-toe, chess, checkers, and connect4. They provided a number of examined states and performed simulations over 5 seconds using a flat MC approach. Additionally, the author tested the version with transposition tables and differentiate results for game stages described as early, middle, and end. Compilation times were not provided.

• (Kissmann and Edelkamp 2010) proposes a process of instantiation, i.e., grounding all occurrences of variables within the GDL description. Instantiated rules, although syntactically larger, are usually much faster to process during the gameplay. (The instantiation algorithms become a subbranch of GDL research, c.f. e.g., (Vittaut and Méhat 2014).)

The experiments were performed to test how many of the 171 games from the Dresden GGP server were possible to ground via a Prolog-based approach, and via their dependency graph technique. Also, for 10 games, the performance of SWI-Prolog reasoner on standard and instantiated game rules was compared using a number of Monte Carlo simulations as a measure.

• (Saffidine and Cazenave 2011) presents a forward chaining GDL compiler into Ocaml. The results compare a number of Flat Monte Carlo playouts performed within 30 seconds against the YAP Prolog interpreter. They contain compilation times and the initial and final file sizes. The test set consists of 9 games, including breakthrough, connect4, and three variants of tictactoe.

• (Kaiser and Stafiniak 2011a)[1] presents a method of rewriting GDL into previously mentioned Toss system, which makes use of CNF and DNF conversions, SAT solvers, and first-order logic model checkers.

The authors compare the efficiency of their system by playing against Fluxplayer (Schiffel and Thielscher 2007), and showing winrates on four games: breakthrough, connect4, connect5, and pawn whopping. The Toss search algorithm was based on constant-depth alpha-beta.

• (Kowalski and Szykuła 2013) presents a highly optimized bottom-up GDL to C++ compiler with multiple features, including predicates flattening and optimized data structures to fasten query times. The results include compilation times, the number of simulations, and the number of computed game states per second during Flat MC. The comparison was made against ECLiPSe Prolog reasoner, and included 6 games: tictactoe, blocker, connect4, breakthrough, checkers, and skirmish.

• (Schofield and Saffidine 2013) is another forward chaining GDL compiler, which is, in some sense, an improved version of the previous work. It contains a more detailed experiment section, measuring compilation times, the number of states per second, and the number of playouts per 30 seconds of Flat MC playouts on 19 games. The results were not directly compared to any baseline.

[1] http://toss.sourceforge.net/

• (Björnsson and Schiffel 2013) attempts to be a complex comparison on existing GDL reasoners. Actually, it compares Fluxplayer (Schiffel and Thielscher 2007) and CadiaPlayer (Björnsson and Finnsson 2009)[2], presenting one test against some basic reasoners (also including GGP Base package (Schreiber 2013)), but omits head-on comparison with any of the previously mentioned published approach.

The results for Fluxplayer and CadiaPlayer include visited nodes per second for two search algorithms: Monte Carlo and Min-Max. There are also tests versus game-specific reasoners. The comparison with other engines does not include numbers, only a bar chart with relative speeds. All experiments were performed on a set of 12 testgames.

• (Świechowski and Mańdziuk 2016) describes various optimizations of game encoding, including rewriting queries and specific memory representation. The experiments were performed on 28 games, and compared a number of random Monte Carlo simulations versus ECLiPSE Prolog and YAP Prolog.

• (Sironi and Winands 2017) introduces optimizations on propositional networks, which are a very efficient alternative representation of a GDL code. Experiments were performed on 13 games, testing multiple variants of the algorithm. The results include the number of nodes per second in flat MC, initialization times, and the size of the resulting network. The work also compares the speed and the win-percentage versus GGP-Base Prover (Schreiber 2013).

Findings

The most conspicuous issue is the almost complete lack of direct comparisons. In our opinion, this is due to the following reasons. First, most of the systems were not available as open-source (partially because of the domain's competitiveness, partially for other reasons like reluctance to share messy code). However, some of them, e.g., CadiaPlayer or Toss, are available for quite a long time, and they are still not used as a testbase.

This is because of the second reason – GGP agents are very complex systems, and using off-the-shelf code causes many problems. They are usually not designed to be operated by anyone except the authors, and sometimes they are even prepared to work on a specific hardware / network architecture. Thus, even with the source, modifying them to ensure fair comparison (e.g., the same search algorithm used) is a very tiresome work. A remedy for these issues would be publishing a full list of project dependencies or a Dockerfile, allowing its fully accurate setup.

On the positive sides, most of the enumerated experiments use a similar comparison approach, i.e., the number of playouts or the number of visited states during Flat Monte Carlo simulations. Also, usually, the papers mention the hardware specifications, of course, with various levels of details. Often, there is missing information about the amount of RAM and operating system, not to mention nearly non-existing detailed, but still important, data about software versions and the number of running threads.

[2] http://cadia.ru.is/wiki/public:cadiaplayer:main#cadiaplayer_source

However, the flat MC algorithm is commonly underspecified. While it is generally understood, it can vary in some subtle details, e.g., whether the final goals of the players are computed after each random playout.

We can observe a common trend for sources of the game descriptions: for the earlier works, it is Dresden GGP Server (Gunther and Schiffel 2013); and for the later ones, GGP Base repository (Schreiber 2016). Providing the location of the source should allow to accurately know the GDL codes used, which is very important as in GDL even small encoding differences may influence the reasoning difficulty. However, as for some games multiple versions of the rules exist in the same repository, the game name should be stated in a form allowing unambiguous matching.

Let us observe that the sets of testgames used in experiments are not standardized in any way. Firstly, they are usually minimal, especially in early works, due to the time required for gathering enough reliable samples. Secondly, the choices of test games are often discordant. Some games are most common than others, but finding a good testset to compare results with other approaches remains a difficult task. Just to mention that the only game commonly tested in all the above works is tictactoe.

The last observation is that with time, the quality of the presented experiments improves. More recent publications tend to be better documented and contain more insightful experiments. However, still many solutions are not released as open-source, so reproduction of the results is not possible.

Comparison to GVGAI

As an example of a newer and better-organized GGP competition, we will briefly describe General Video Game AI, launched in 2014, and running multiple-track competitions since then (Perez-Liebana et al. 2019a). With the downside of forcing agents to be written in Java or Python only, GVGAI standardizes the agent's structure and communication protocol more firmly. There is no net-based communication. To take part in the competition, the agent's properly formatted sources have to be sent to the competition page, where they are automatically run and evaluated.

The framework itself is open-sourced and regularly updated. Thus, one official game repository is available, every game is labeled, and its rules are corrected if necessary. Players' agents are not necessarily open, however some data about their performance, e.g., score on each of the global sets of testing games, is publicly available on the website.

Thus, although still far from the ideal, the example of GVGAI shows that it is possible to organize a general game playing competition that will allow easier data exchange, more insights into the results obtained by the agents, and thus better reproducibility.

Comparing Different GGP Formalisms

The existence of multiple GGP languages arises a natural set of questions: which language is better – more universal, more efficient, more readable, easier for learning or PCG; do agents in one system have some natural advantage over the agents from the other system; can we automatically translate rules between these languages, etc.? Because obtaining any reliable answer to these questions is very difficult and requires much work, both conceptually and programmatically, there is relatively little research tackling these problems.

Syntactical translations

Automatic translation of game rules from one GGP language to the other, of course, depends on the generality of those languages. So far, apart from the theoretical mapping of formalisms like in (Thielscher 2011), only one-sided translations from simple GGP languages to versions of Stanford's GDL were proposed. All works described here present experiments in which translated rules are fully playable games (in either GDL or Toss).

• First published translation concerns mapping from GDL to Toss (Kaiser and Stafiniak 2011b). The authors provide a detailed description of the translation, with multiple informative examples for each step, and formally prove its correctness. The obtained Toss rules are described as ,,inefficient" and ,,verbose", compared to the ones created manually, but no numbers are provided. A brief experimental section contains the results on matches between Toss players using manual and translated rules on two games only: one ended with 100% of ties, the other one with 10% advantage for a manually written description.

• In (Kowalski 2014), the author describes a mapping from very domain-specific Card Game Description Language (Font et al. 2013) into GDL-II (GDL extension with randomness and imperfect information). The work puts an overview of the translation rules, proves its correctness, and measures dependence between the complexity of both descriptions. Experiments are performed on three game codes (plus some variants) that were originally published for the card language. The main conclusion was aimed at the verbosity of GDL, as using the proposed translation, even a very simple game, requires hundreds of rules to be encoded. The system is not open-sourced, nor any more detailed analysis of the translation is available.

• On the other hand, (Sutowicz 2016) presents a translation from Simplified Boardgames (Björnsson 2012) into standard GDL, which aims at producing code that is computationally optimized. In particular, when translating regular expressions into GDL rules, it tries to share partial expressions to reduce code redundancy. The thesis contains proofs of bounds: on the size of resultant code and on the time complexity of the algorithm. The experiments show the improvement factors of applied optimizations. Tests have been performed on 11 chess-like games, including procedurally-generated ones and specially handcrafted examples. What is worth mentioning, although without documentation, the translator source is available online[3].

Game playing comparison

Yet another, even more challenging and definitely more error-prone task is to experimentally assess language properties via the agent's performance when both agents belong to different GGP systems.

[3] https://github.com/uicus/sbg2gdl

• In previously mentioned Toss publication (Kaiser and Stafiniak 2011a), the authors, to show the benefits of their formalism, defined several board games in it and created move translation scripts allowing playing against GGP agents. Thus, the comparison works under the assumption that each game is encoded optimally for both languages. (Which gives some advantage for Toss, where it is easier to derive game heuristics from the game rules automatically.)

Experiments were performed between two different search algorithms, Toss and Fluxplayer, run on different and unknown hardware on four games – with one clear win for Toss, one for Fluxplayer, and two complete ties. So, apart from that, the presented solution is „good enough to win against a state of the art GGP player", any more insightful conclusions are impossible to make.

• A very similar approach, aiming to assess the progress of Stanford's GGP by comparing state-of-the-art players with an exemplary agent from much simpler Simplified Boardgames class, has been taken in (Kowalski and Kisielewicz 2015). Here, the authors developed a bridge, allowing simplified boardgames agent to play using Stanford's GGP competition protocol.

A simple min-max player with evaluation function obtained by temporal difference learning was paired against two GGP players, including 2014 IGGPC champion Sancho, on four chess-like games – however, all three agents were launched on different hardware. The interesting aspect of the experiment was that in Simplified Boardgames the agent is aware of exiting pieces and board, thus it can reason about their values. In Stanford's GGP, those concepts are not so clearly visible, so in multiple tests, the simple player easily won against these advanced GDL-based agents.

• Regular Boardgames is yet another GGP language that proves to be faster than GDL (Kowalski et al. 2019b). To visualize the advancement, the authors matched the rules of 12 games in both languages and performed experiments using both perft (computing the whole game tree to a fixed depth) and Flat MC counting the number of visited states. Two implementations of GDL engine were used: ECLiPSe Prolog (withing the gamechecker tool), and propnets from (Sironi and Winands 2017). The system is available as open-source with all the game codes used for the experiment[4].

Case study: comparing the efficiency of Ludii and Regular Boardgames

The last example that we describe is a recent comparison (Piette et al. 2019b; 2019a) of three different GGP languages (in particular, the efficiency of reasoning): Ludii, Regular Boardgames, and GDL. The series of works is recent and leads to some interesting conclusions regarding conducting and reproducing experimental research. We performed a detailed analysis of the experiments, trying to reproduce them.

It turned out that they are a good visualization of possible issues when comparing general game playing systems and also of difficulties coming out during a reproduction attempt. We highlight some difficulties in methodologies and carried experiments that can significantly distort the results and actually turn the conclusions into the opposite. This example is yet another evidence of how important it is, for any sort of experimental justification, allowing its easy and undoubtful repetitiveness. We present this analysis in the hope of avoiding similar problems in any further research of this kind, allowing comparisons as fair as possible, which do not cause the need for questioning the results.

Reproduction attempt Because of the unavailability of the Ludii version used in (Piette et al. 2019b), we performed an analysis based on one of the publicly released (and newer) Ludii version.

When we took Ludii games with the corresponding names according to the benchmark and analyzed their rules, we found out that only 5 out of 14 games have fully equivalent rules to those existing in RBG 1.0. In most of the remaining cases, when the rules embed a different game variant than that existing in RBG 1.0, we made an attempt to reimplement that variant with more corresponding rules to the version in Ludii (but still only to some practicable extent). Then we performed our benchmark. A comparison of the results from both experiments is shown in Table 1. For the first two games, we visualize the problems caused by a mismatch between games that differ only by a variation of rules. Our detailed technical analysis of this study is available at (Kowalski et al. 2019a).

Based on our analysis of the benchmark in (Piette et al. 2019b), we have discovered several schemes that could occur when attempting cross-language comparison. Here, we mention three most important of our findings.

1. First, what influenced the results the most is that the majority of the compared games do not have the same rules in all the three GGP systems. As we do not know the rules used in the previous Ludii's version, we base on the knowledge that correct rules of the concerned games in a proper variant, fully corresponding to those in RBG 1.0, were not present in public Ludii versions that soon followed the paper, and in most cases they are still not available. The differences in the other games were usually based on simplifying the computation in favor of Ludii.
2. The repeated experiment for the RBG part, using exactly the same code, in most cases gave similar results, provided the hardware differences. However, there are two exceptions, i.e., connect-4 and reversi, where the reported results were respectively about 2 and 4 times smaller than we could expect in our benchmark.
3. The results for GDL were obtained on different hardware than those for RBG and Ludii. Indeed, exactly the same GDL results were reported before in (Piette et al. 2019a), where they were obtained with a slower processor, and the performance differences are also visible in our results. Although the hardware used for GDL was not directly specified, we definitely see putting these results in the same table with the others to be misleading. Furthermore, the result for chess was produced using the GGP-Prover instead of a propnet, despite that there was available an efficient chess implementation working well under a propnet.

We note that some of the above-mentioned problems have

[4]https://github.com/marekesz/rbg/

Table 1: The original and reproduced results of the efficiency of reasoning in RBG, Ludii, and GDL for the **flat Monte Carlo** test. The values are the **numbers of playouts per second**.

Results from (Piette et al. 2019b)				Results from (Kowalski et al. 2019a)			
Game	**RBG 1.0**	**Ludii**	**GDL**	**Game**	**RBG 1.0/1.0.1**	**Ludii 0.3.0**	**GDL propnet**
Amazons	625	4,349	185	Amazons-orthodox	569	*n/a*	4
				Amazons-split	8,798	3,859	365
Arimaa	0.11	714	*n/a*	Arimaa-orthodox	0.14	*n/a*	*n/a*
				Arimaa-split	666*	446†	*n/a*
Breakthrough	16,694	4,741	1,123	Breakthrough	19,916	3,546	2,735
Chess	714	720	0.06	Chess-fifty move	523*	14†	45
Connect-4	84,124	94,077	13,664	Connect-4	190,171	63,427	45,894
English draughts	14,262	8,135	872	English draughts-split	23,361*	7,111†	3,466
Gomoku	2,212	42,985	927	Gomoku-free style	2,430*	26,878	*n/a*
Hex	5,787	11,077	*n/a*	Hex	6,794	10,625	*n/a*
Reversi	2,012	2,081	203	Reversi	8,682	1,312	373
The mill game	7,423	72,734	*n/a*	The mill game	10,102*	2,467†	*n/a*
Tic-tac-toe	400,000	535,294	85,319	Tic-tac-toe	526,930	422,836	104,500

* This game code was not originally available in RBG 1.0 and was added later.
† The rules in Ludii differ from the others.

been fixed in a more recent update[5], with better matching of games and more accurate computation of RBG. Sadly, some of the issues pointed out in (Kowalski et al. 2019a) still apply. Nevertheless, indeed, the conclusions drawn from the newer experiment are in the opposite to those from the older work and mostly agree with those from our tests.

Finally, we note that in (Piette et al. 2019b), there is also a comparison of game descriptions between Ludii, RBG, and GDL, which aims to estimate the clarity and simplicity of those languages. The number of tokens (as a measure of description complexity) was calculated for each language. However, the method of calculating is unclear enough, so that we were not able to find out the algorithm. It has not been stated anywhere in the text, and all straightforward approaches that we tried to reproduce the method resulted in different numbers.

Findings

As for this branch of GGP research there is even less common ground between various approaches, there is also harder to find solid connections between each paper that will somehow force maintaining some standards.

Early works can be characterized by extremely small experimental sections, providing only a few results, usually not documented in detail. The trivial reason, being somewhat a justification, is that it is the consequence of the other authors' papers on which they based on. When a GGP language is introduced providing only 3 example of games with no more games to be found at any available repository, any approach to work with that language requires either to codify new games in it or to stick with the existing ones only. This also leads to the next question – is the primary work descriptive enough so that we can define our own games? What if it does not provide gamechecker, etc.? Sadly, such situations happen, but fortunately, there is definitely less of them as time goes by.

When comparing game-playing algorithms or the efficiency of search engines, obviously, information about the system specifications and the algorithm used are crucial. But the mentioned research reveals a more subtle cause that can have a tremendous impact on the final results – a proper game matching. What is usually not so important from the human point of view when just thinking about the game may be very important from the algorithmic perspective. For many well-known games, it is surprisingly hard to specify what the „standard" rules actually are. Under the same name, often many rule variations can be hidden. Obviously, the compared games should be the same, while what does it mean may be already not so obvious. We think that the most natural definition of "being the same" could be "have isomorphic game trees", because then there exists a straightforward translation between playing these games. However, specifying the rules or referring to exact implementations is still a rare practice. Most of the research use common names without specifying the details, or even change the underlying rules of the game from one publication to another, yet still refer to it as it would have the same rules.

Finally, a commonsense assumption is that for both languages the game is encoded in an optimal way. Thus, when multiple game versions are available in the repository, the best-performing one should be used, and furthermore, using the best-known algorithm.

Conclusion

Based on the findings, we provide the following recommendations with explanations. We hope that they will positively

[5]http://ludeme.eu/outputs/AAAI_20-6.pdf

affect communication and cooperation in the community, and help to produce high-quality research on which further developments can be built collaboratively and safely.

Of course, apart from our domain-based conclusions, all the standard good research habits apply, as making source code openly available and well documented, or sharing easy-to-run test package to allow independent result reproduction, including more detailed recommendations that can be found in (Gundersen, Gil, and Aha 2018).

(1) Provide the exact game descriptions that were used. Particularly in the cases when games in GDL were compared, usually only the name of the game was provided, despite the existence of various substantially different implementations.

(2) When comparing different game descriptions of the same game, make sure that they are equivalent. In the best case, provide a definition (e.g., by the same games, we understand those with isomorphic game trees). Testing whether a game description is correct appears to be a difficult task. Errors often occur in all GGP systems, sometimes left unnoticed for years.[6] To test the correctness, we propose commonly performing two kinds of automatic tests: **perft**, computing the number of all legal playouts up to a fixed depth, and **flat MC**, which provides statistics like the average goals, depth, and moves. Furthermore, for popular games, the values could be published to allow verification of the correctness of new implementations (e.g., the perft results for chess are available at oeis[7]).

(3) Choose the testset that maximally covers the ones used in correlated research. Given limited time and resources, the authors usually prioritize examples for which they have the best results or the results of which give the best overview of their method. However, we advise to also take into account testcases that were considered in earlier works, as it sets the results more in context and helps followers to better estimate their own work compared to what is already existing.

(4) Provide the system specification and key software. This applies to the operating system and compiler or JVM versions.

(5) Run the test on a maximally idle system. It depends on the particular test. It is well known that processes in the background can influence the efficiency of a computation. Hence, to make the results reliable and maximally repetitive, it is better to avoid any background work.

(6) If the test should run on only one core, force this additionally to avoid misuse. In some cases, a single-threaded program does not really imply that only one core is used. This is common in modern Java, which, e.g., runs garbage collection on separate threads. Since the real load of the system is then larger, it can cause unfair comparison with native programs that truly are executed on one core. On linux systems, enforcing a one-core run can be achieved through `taskset` command. Surprisingly, this can cause Java programs running faster and consuming less memory, as well as slower, depending on the test. In any case, a note of whether the one-core restriction was used should be given, to avoid miscomparison when reproduced.

(7) If possible, when comparing with an existing implementation, ask the authors to ensure the correct usage. Although this may sound childish, it is still applicable as concerns surprisingly many cases – from being unsure of an answer received, up to the desire of not revealing own plans and intentions regarding other's work. Irrespective of the reason, avoiding such contact reduces the quality of the conducted research. Many systems are not documented well enough to ensure its proper usage, guaranteeing, e.g., their maximum performance.

[6]Recently, we have found a subtle bug in all existing GDL implementations of the amazons game, commonly used in many experiments. A fixed version was proposed.

[7]https://oeis.org/A048987

Acknowledgments

We thank Chiara F. Sironi for sharing the GDL propnet code and for helping with using it. This work was supported by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

References

Abiteboul, S.; Hull, R.; and Vianu, V., eds. 1995. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., 1st edition.

Björnsson, Y., and Finnsson, H. 2009. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1):4–15.

Björnsson, Y., and Schiffel, S. 2013. Comparison of GDL Reasoners. In *GIGA*, 55–62.

Björnsson, Y. 2012. Learning Rules of Simplified Boardgames by Observing. In *European Conference on Artificial Intelligence*, volume 242 of *FAIA*. 175–180.

Browne, C., and Maire, F. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1):1–16.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Campbell, M.; Hoane, A. J.; and Hsu, F. 2002. Deep Blue. *Artificial intelligence* 134(1):57–83.

Font, J. M.; Mahlmann, T.; Manrique, D.; and Togelius, J. 2013. A Card Game Description Language. In *Applications of Evolutionary Computation*, volume 7835 of *LNCS*. 254–263.

Genesereth, M., and Thielscher, M. 2014. *General Game Playing*. Morgan & Claypool.

Genesereth, M.; Love, N.; and Pell, B. 2005. General Game Playing: Overview of the AAAI Competition. *AI Magazine* 26:62–72.

Gundersen, O. E.; Gil, Y.; and Aha, D. W. 2018. On reproducible ai: Towards reproducible research, open science, and digital scholarship in ai publications. *AI Magazine* 39(3):56–68.

Gunther, M., and Schiffel, S. 2013. Dresden General Game Playing Server. http://ggpserver.general-game-playing.de.

Kaiser, Ł., and Stafiniak, Ł. 2011a. First-Order Logic with Counting for General Game Playing. In *AAAI*, 791–796.

Kaiser, Ł., and Stafiniak, Ł. 2011b. Translating the Game Description Langauge to Toss. In *GIGA*, 91–98.

Kissmann, P., and Edelkamp, S. 2010. Instantiating General Games Using Prolog or Dependency Graphs. In *KI 2010: Advances in Artificial Intelligence*, volume 6359 of *LNCS*. 255–262.

Kowalski, J., and Kisielewicz, A. 2015. Testing General Game Players Against a Simplified Boardgames Player Using Temporal-difference Learning. In *IEEE Congress on Evolutionary Computation*, volume 9597, 1466–1473.

Kowalski, J., and Kisielewicz, A. 2016. Towards a Real-time Game Description Language. In *International Conference on Agents and Artificial Intelligence*, volume 2, 494–499.

Kowalski, J., and Szykuła, M. 2013. Game Description Language Compiler Construction. In *AI 2013: Advances in Artificial Intelligence*, volume 8272 of *LNCS*. 234–245.

Kowalski, J., and Szykuła, M. 2016. Evolving Chesslike Games Using Relative Algorithm Performance Profiles. In *EvoApplications 2017: Applications of Evolutionary Computation*, volume 9597 of *LNCS*. 574–589.

Kowalski, J.; Mika, M.; Sutowicz, J.; and Szykuła, M. 2019a. A note on the empirical comparison of RBG and Ludii. *arXiv preprint arXiv:1910.00309*.

Kowalski, J.; Mika, M.; Sutowicz, J.; and Szykuła, M. 2019b. Regular Boardgames. In *AAAI*, 1699–1706.

Kowalski, J. 2014. Embedding a Card Game Language into a General Game Playing Language. In *Proceedings of the 7th European Starting AI Researcher Symposium*, 161–170.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General Game Playing: Game Description Language Specification. Technical report, Stanford Logic Group.

Newell, A.; Shaw, J. C.; and Simon, H. A. 1959. Report on a general problem solving program. In *IFIP congress*, volume 256, 64.

Pell, B. 1992. METAGAME in Symmetric Chess-Like Games. In *Heuristic Programming in Artificial Intelligence: The Third Computer Olympiad.*

Perez-Liebana, D.; Liu, J.; Abdel Samea Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2019a. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *IEEE Transactions on Games*.

Perez-Liebana, D.; Lucas, S. M.; Gaina, R. D.; Togelius, J.; Khalifa, A.; and Liu, J. 2019b. *General Video Game Artificial Intelligence*. Morgan and Claypool Publishers. https://gaigresearch.github.io/gvgaibook/.

Piette, E.; Soemers, D. J.; Stephenson, M.; Sironi, C. F.; Winands, M. H. M.; and Browne, C. 2019a. Ludii-the ludemic general game system. *arXiv preprint arXiv:1905.05013*.

Piette, E.; Stephenson, M.; Soemers, D. J.; and Browne, C. 2019b. An Empirical Evaluation of Two General Game Systems: Ludii and RBG. In *IEEE Conference on Games*.

Pitrat, J. 1968. Realization of a general game-playing program. In *IFIP Congress*, 1570–1574.

Saffidine, A., and Cazenave, T. 2011. A Forward Chaining Based Game Description Language Compiler. In *GIGA*, 69–75.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *Science* 317(5844):1518–1522.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A Successful General Game Player. In *AAAI*, 1191–1196.

Schofield, M., and Saffidine, A. 2013. High Speed Forward Chaining for General Game Playing. In *GIGA*, 31–38.

Schreiber, S. 2013. The General Game Playing Base Package. https://github.com/ggp-org/.

Schreiber, S. 2016. Games – base repository. http://games.ggp.org/base/.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484–503.

Sironi, C. F., and Winands, M. H. M. 2017. Optimizing Propositional Networks. In *Computer Games*. Springer. 133–151.

Siwek, C.; Kowalski, J.; Sironi, C. F.; and Winands, M. H. M. 2018. Implementing Propositional Networks on FPGA. In *AI 2018: Advances in Artificial Intelligence*, volume 11320 of *LNCS*. 133–145.

Sutowicz, J. 2016. Simplified Boardgames to Game Description Language Translator (in Polish). Bachelor's Thesis.

Świechowski, M., and Mańdziuk, J. 2016. Fast interpreter for logical reasoning in general game playing. *Journal of Logic and Computation* 26(5):1697–1727.

Thielscher, M. 2010. A General Game Description Language for Incomplete Information Games. In *AAAI*, 994–999.

Thielscher, M. 2011. *Translating General Game Descriptions into an Action Language*, volume 6565 of *LNCS*. 300–314.

Thielscher, M. 2017. GDL-III: A Description Language for Epistemic General Game Playing. In *IJCAI*, 1276–1282.

Vittaut, J.-N., and Méhat, J. 2014. Efficient Grounding of Game Descriptions with Tabling. In *Computer Games*, volume 504 of *CCIS*. 105–118.

Waugh, K. 2009. Faster State Manipulation in General Games using Generated Code. In *GIGA*.