Ludii and XCSP: Playing and Solving Logic Puzzles

Cédric Piette
Centre de Recherche en Informatique de Lens
Université d'Artois
Lens, France
piette@cril.fr

Éric Piette, Matthew Stephenson, Dennis J.N.J. Soemers, Cameron Browne
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, the Netherlands
{eric.piette,matthew.stephenson,dennis.soemers, cameron.browne}@maastrichtuniversity.nl

***Abstract*—Many of the famous single-player games, commonly called puzzles, can be shown to be NP-Complete. Indeed, this class of complexity contains hundreds of puzzles, since people particularly appreciate completing an intractable puzzle, such as Sudoku, but also enjoy the ability to check their solution easily once it's done. For this reason, using constraint programming is naturally suited to solve them. In this paper, we focus on logic puzzles described in the Ludii general game system and we propose using the XCSP formalism in order to solve them with any CSP solver.**

***Index Terms*—Knowledge Representation, Constraint Programming, General Game AI.**

I. Introduction

Since the Nikoli company[1] publishes many puzzles in different newspapers, such as the well-known Sudoku, solving pure deduction puzzles (mainly Japanese logic puzzles [1]) became a widespread pastime around the world.

In the context of General Game Playing (GGP), where artificial agents have to be capable of playing a wide variety of games (including puzzles) [2], Monte Carlo Tree Search (MCTS) [3] is now considered as one of the best approaches in the absence of domain specific knowledge [4]. For puzzles, two main previous works on MCTS exist. The first one, called Single Player MCTS used in the *SameGame* player has been introduced in [5]. This variant adds a new term in the UCB formula, and uses a heuristically guided default policy for simulations in order to optimize their resolution. The second work defines an adapted UCT algorithm for the combinatorial optimisation problem of feature selection [6].

However, these two variants – and MCTS in general – do not tend to perform well on pure deduction puzzles. The nature of these puzzles [7] may not be suitable for this paradigm, for instance due to most of them exhibiting only a single solution.

Regarding complexity, many of the (logic) puzzles, are shown to be NP-complete [8]. In this context, constraint programming (CP) appears to be a particularly effective means of solving them [9], [10]. Moreover, CP is well-studied, leads to highly explainable solutions, and finding solutions is typically efficient [11]. In this paper, we present how the General Game system Ludii [12] can model and play any logic puzzle through the XCSP formalism [13], enabling the use of any compatible state-of-the-art solver to efficiently solve them.

[1]Nikoli: nikoli.co.jp/en/puzzles/

II. Ludii and XCSP

In this section, we briefly describe Ludii and XCSP, as well as the translation process from Ludii to XCSP for puzzles.

A. Ludii

Within the context of the Digital Ludeme Project[2] [14], a new general game system called Ludii was recently proposed. This system is based on a ludemic modelisation and a class grammar approach for games [15].

In Ludii, a game is given by a 3-tuple of ludemes $\mathcal{G} = \langle Mode, Equipment, Rules \rangle$. $Mode$ denotes a finite set of k deterministic players. $Equipment = \langle C^t, C^p \rangle$ denotes a set of containers C^t, and a set of components C^p. Finally, $Rules$ defines the operations of the game, which is split in three distinct parts: $start$, $play$, and end.

In contrast to the standard General Game system using the Game Description Language (GDL) [16], the ludemic approach offers the capability to model puzzle rules. Each ludeme encapsulates the concepts commonly used like arithmetic operators, inequality between many grid cells or specific regions (row, column, etc). An example Ludii description for a Sudoku puzzle on a 4$\times$4 grid is provided in Figure 1, with the puzzle itself depicted in Figure 2.

Ludii provides many benefits relative to GDL. Among them, the description of the games are self-explanatory to non-specialist readers and all games available in the system can be played by both humans and/or AI. Moreover, even if Ludii uses MCTS as the core method for AI move planning, the structural composition of the games, and more particularly puzzles, can be exploited to make use of different paradigms such as CP.

B. XCSP

XCSP3 [13] is a recent format to build integrated representations of combinatorial constrained problems. This new format is able to deal with mono/multi optimisation, many types of variables (integer, symbolic, graph, set, multi-set,

[2]Digital Ludeme Project: http://www.ludeme.eu/

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

```
(game "Sudoku 4x4"
 (mode 1)

 (equipment {
  (SudokuBoard 2)
  (number P1 {1 2 3 4})
  })

 (rules
  (start {
   (place
    {4 1 3 3  1}
    {1 5 7 13 15}
   )
  })

 (play (to {1 2 3 4} (empty)))

  (if (equal (count (empty)) 0)
   (end
    (if (and {
     (allDifferent (Row 0))
     (allDifferent (Row 1))
     (allDifferent (Row 2))
     (allDifferent (Row 3))
     (allDifferent (Column 0))
     (allDifferent (Column 1))
     (allDifferent (Column 2))
     (allDifferent (Column 3))
     (allDifferent (set {0  1  4  5}))
     (allDifferent (set {2  3  6  7}))
     (allDifferent (set {8  9  12 13}))
     (allDifferent (set {10 11 14 15}))
     })
     (result 1 Win)
     (result 1 Loss)
    )
   )
  )
 )
)
```

Fig. 1. Game description of a Sudoku puzzle on a 4×4 grid in Ludii.

	3		1
	1		3
	4		

Fig. 2. Example of a Sudoku puzzle on a 4×4 grid.

etc.), cost functions, reification, views, annotations, variable quantification, distributed, probabilistic and qualitative reasoning. The new format is made compact, highly readable, and easy to parse. Similar to the philosophy of ludemes this format allows us to encapsulate the structure of the problem models, through the possibilities of declaring arrays of variables, and identifying syntactic and semantic groups of constraints. A competition for solvers using the XCSP format is organised annually,[3] and as a result of this, most of the major constraint solvers developed by the CP community support XCSP. Accordingly, an XCSP instance can be solved using many different efficient solvers, including `Abscon`, `Choco`, `Oscar` and `SAT4J` to name of few. A website (xcsp.org) is

[3]XCSP competition: http://www.xcsp.org/competition

```
<instance format="XCSP3" type="CSP">
<variables>
  <array id="x" size="[4][4]"> 1..4 </array>
</variables>
<constraints>

<instantiation class="hints">
    <list> x[0][1] x[1][1] x[1][3] x[3][1] x[3][3]</list>
    <values> 4 1 3 3 1 </values>
</instantiation>

<group>
  <allDifferent> %... </allDifferent>
    <args> x[0][] </args>
    <args> x[1][] </args>
    <args> x[2][] </args>
    <args> x[3][] </args>

    <args> x[][0] </args>
    <args> x[][1] </args>
    <args> x[][2] </args>
    <args> x[][3] </args>

    <args> x[0..1][0..1] </args>
    <args> x[0..1][2..3] </args>
    <args> x[2..3][0..1] </args>
    <args> x[2..3][2..3] </args>
</group>
</constraints>
</instance>
```

Fig. 3. Game description of a Sudoku puzzle on a 4×4 grid with XCSP.

```
<instantiation id='sol1' type='solution'>
    <list> x[] </list>
    <values>
        3 4 1 2 2 1 4 3 1 2 3 4 4 3 2 1
    </values>
</instantiation>
```

Fig. 4. Solution of the XCSP instance in Figure 3.

developed conjointly with the format, which contains many models and series of instances, as well as different tools such as parsers in Java and C++.

C. From Ludii to XCSP

A Constraint Satisfaction Problem (CSP) consists of a set of variables – each associated with a domain of possible values – and a set of constraints that link the variables and define allowed combinations of values among them.

Constraints can have several forms: two basic forms of them are (i) enumerating the list of allowed/forbidden tuples between variables and (ii) using common simple constraints in intention such as $=, \neq, <, \leq$, etc. Moreover, since the mid 90's, the concept of so-called *global constraints* has been introduced. Such constraints aim to improve the succinctness of constrained structures present in different problems, and are associated with more powerful filtering algorithms that can take into account the specificity of the formulated constraint to further reduce the domains of the variables, boosting the subsequent search for a solution. The first introduced and most famous global constraint is *allDifferent*, which states that each variable in its scope must take values different from all other ones.

Interestingly, logic puzzles very often exhibit structure that can be encoded into global constraints. Furthermore, in Ludii, the library provides some ludemes similar to the main global constraints. For example, the global constraint *allDifferent* is also used as a ludeme in Ludii (e.g. Figure 1), making the translation process easier.

Thanks to this proximity between the two languages, a translation process from Ludii to XCSP is possible in order to use the solution of the CSP problem generated as a sequence of moves for the Ludii game. Note that our study is restricted to one-player games ($Mode = \{1\}$) using a single container ($|C^t| = 1$).

The variables and their domains are extracted from the $Equipment$ of a Ludii Game $\mathcal{G}$. Each CSP variable v is generated from each grid cell of the container. The domain of each variable corresponds to the set C^p.

The generation of the constraints is obtained from the $Rules$ of $\mathcal{G}$. However, for the initial state ($start$ rules) and the terminal state (end rules), the process is different. For puzzles, only the ludeme $place$ is used to put each component on different grid cells. This ludeme can easily be translated into constraints that assign the initial values to the corresponding variables. For the terminal state, each ludeme associated with a global constraint in the XCSP formalism is translated directly using the subset of variables corresponding to the region define on the ludeme. As an example, the translation of the $4{\times}4$ Sudoku described in Figure 1 is given in Figure 3.

In this example, the lines 7 to 10 correspond to the $start$ rules in $\mathcal{G}$, and the lines between 12 to 28 correspond to the end rules, each of them generated from the ludeme *allDifferent* on different regions.

The solution provided by the CSP solver is translated into a sequence of Moves for the Ludii system. For each assignment, if the value associated is different to 0 (corresponding to an empty cell) and if in the $start$ rules no component is placed in the grid cell corresponding to the variable assigned, we apply a move to add the component corresponding to the value to the grid cell.

As an example, the CSP solution to the XCSP instance is given in Figure 4. The corresponding sequence of moves is: `Add(0,3)`, `Add(2,1)`, `Add(3,2)`, `Add(4,2)`, `Add(6,4)`, `Add(8,1)`, `Add(9,2)`, `Add(10,3)`, `Add(11,4)`, `Add(12,4)`, `Add(14,2)`.

III. Experiments

This section describes a number of experiments on different puzzles modelled with the ludemic approach.

A. Setup

In this section, we experiment with the translation and solving processes on some logic puzzles: Futoshiki, Latin Square, Magic Square, N Queens problems, Nonogram and Sudoku. We provide the time used for each process and the size of the XCSP instance obtained by the translation process when given the number of variables, the size of the domain of each variable and the number of constraints. All experiments were conducted on a single core of an Intel(R) Core(TM) i7-8650U CPU @ 1.90 GHz, 2112 MHz with 16GB RAM. To solve the XCSP instance, we use an open-source Java-written Constraint Solver called `Abscon`, mainly developed by Christophe Lecoutre. Ludii and `Abscon` are running with the Java SE Development Kit 11.

B. Results

First, as illustrated by Table I, translations from Ludii to XCSP exhibit negligible runtimes, since they never exceed 1 second. We did not report them in a detailed manner due to lack of space, but we have tried larger sizes of puzzles and converting data towards XCSP formalism does not appear to be a barrier to this approach.

For the solving part, we have deliberately tested games with different sizes, in order to get an idea of the practical limits of the approach. For example, converting and solving Latin Square (size 100) requires more than a minute, which can be seen as too long for some applications. However, all puzzles whose size makes them reasonably doable by humans are entirely solved within a few seconds : N Queens (sizes 4-8), Nonogram (sizes 5-32), Sudoku (sizes 9-25), etc.

C. Discussion

These results allow us to apply online dynamic applications for all “human-sized” puzzles. For instance, Ludii can propose a series of (potentially randomly generated) puzzles and provide help, or prevent the user from mistakes, in a dynamic way through the use of CP solvers.

We intend to use Ludii to model the full range of Nikoli puzzles in a close future, and the translation proposed here is generic; once a new (NP-complete) puzzle is modelled in Ludii, it can be supported by a CP solver for the previous proposed applications (provide help and/or prevent mistakes), without requiring additional implementation work in most cases.

It is important to keep in mind that we “only” propose to translate directly a puzzle from Ludii to an XCSP instance. Here no optimisation approaches such as heuristics or symmetries are used. This first approach only takes advantage of the global constraints.

Consequently, a promising future work would be to include some data for the ludemes used to describe the puzzles in order to make it easier to apply some optimisations commonly used in CSP. As an example, the current direct translation for the Knight’s Tour ($8{\times}8$) on XCSP is done in 2 seconds. However `Abscon` requires 123 seconds to solve it. Handcrafting simple symmetries in the instance reduces the solving time to 19 seconds.

IV. Conclusion and Future Work

This first work is a good example of cross-fertilisation. Indeed, while the Ludii system is now able to make use of decades of algorithmic progresses in Constraint Programming, just by stating puzzles, the XCSP ecosystem enriches its

TABLE I
RESULTS TO GENERATE AND SOLVE PUZZLES WITH LUDII, XCSP AND Abscon (TIME IN SECONDS)

Game	Board Size	Ludii to XCSP in	#Variables	Domain Size	#Constraints	Solved in
Futoshiki	$4{\times}4$	0.301	16	4	12	2.437
	$5{\times}5$	0.303	25	5	21	2.640
	$6{\times}6$	0.311	36	6	22	2.671
	$9{\times}9$	0.341	81	9	58	2.718
Latin Square	$5{\times}5$	0.010	9	5	10	2.265
	$10{\times}10$	0.017	100	10	20	2.531
	$100{\times}100$	0.121	10,000	100	200	142.377
Magic Square	$3{\times}3$	0.012	9	9	8	2.421
	$5{\times}5$	0.013	25	25	12	2.656
	$7{\times}7$	0.015	49	49	16	3.406
N Queens	$4{\times}4$	0.011	16	2	61	3.125
	$8{\times}8$	0.011	64	2	255	4.002
Nonogram	$5{\times}5$	0.013	25	2	10	1.328
	$10{\times}10$	0.013	100	2	20	1.654
	$20{\times}20$	0.014	400	2	40	1.843
	$32{\times}32$	0.015	1,024	2	64	2.656
Sudoku	$9{\times}9$	0.010	81	9	27	2.421
	$16{\times}16$	0.012	256	16	48	2.734
	$25{\times}25$	0.014	625	25	75	3.127

content of instances by providing new benchmarks for the CP community.

In the future, the approach applied here on NP-complete puzzles could be extended to P-SPACE ones, such as Sokoban [17]. In addition, this work paves the way for a potential application of Constraint Programming on any multi-player games modelled on Ludii, in a similar spirit to WoodStock [18] – a constraint-based approach for General Game Playing, and currently the best GGP agent, only compatible with GDL.

As another working track, a related work on [19], proposes another approach called Deductive Search to solve logic puzzles. This approach is a breadth-first, depth-limited propagation scheme for the constraint-based solution of deduction puzzles, using simple logic operations found in standard constraint satisfaction solvers. This approach is particularly efficient for puzzles and has to be compared with the best CSP solvers in order to improve the resolution to any puzzle available on Ludii.

ACKNOWLEDGMENT.

This research is part of the European Research Council-funded Digital Ludeme Project (ERC Consolidator Grant #771292) run by Cameron Browne at Maastricht University's Department of Data Science and Knowledge Engineering.

REFERENCES

[1] H. Collins, *The Times Japanese Logic Puzzles: Hitori Hashi, Slitherlink and Mosaic*, London, 2006.

[2] J. Pitrat, "Realization of a general game-playing program." in *IFIP Congress*, 1968, pp. 1570–1574.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–49, 2012.

[4] H. Finnsson and Y. Björnsson, "Learning simulation control in general game-playing agents," in *Proceedings of AAAI'10*, 2010, pp. 954–959.

[5] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, "Single-player monte-carlo tree search for samegame," *Know.-Based Syst.*, vol. 34, pp. 3–11, Oct. 2012.

[6] R. Gaudel and M. Sebag, "Feature selection as a one-player game," in *Proceedings of ICML'10*, 2010, pp. 359–366.

[7] C. B. Browne, "The nature of puzzles," *Game & Puzzle Design*, vol. 1, no. 1, pp. 23–34, 2015.

[8] D. M. Costa, "Computational complexity of games and puzzles," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1807.04724

[9] B. O'Sullivan and J. Horan, "Generating and solving logic puzzles through constraint satisfaction," in *Proceedings of AAAI'07, Canada*, 2007, pp. 1974–1975.

[10] M. Çelik, H. Erdogan, F. Tahaoglu, T. Uras, and E. Erdem, "Comparing ASP and CP on four grid puzzles," in *Proceedings of the 16th International RCRA Workshop: Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2009.

[11] J. Jaffar and M. J. Maher, "Constraint logic programming: A survey," *J. Log. Program.*, vol. 19/20, pp. 503–581, 1994.

[12] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne, "Ludii - the ludemic general game system," *CoRR*, vol. abs/1905.05013, 2019.

[13] F. Boussemart, C. Lecoutre, and C. Piette, "XCSP3: an integrated format for benchmarking combinatorial constrained problems," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1611.03398

[14] C. Browne, "Modern techniques for ancient games," in *IEEE Conference on CIG*. IEEE, 2018, pp. 490–497.

[15] C. B. Browne, "A class grammar for general games," in *Advances in Computer Games*, ser. LNCS, vol. 10068, 2016, pp. 167–182.

[16] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008.

[17] R. A. Hearn and E. D. Demaine, "Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *CoRR*, 2002.

[18] F. Koriche, S. Lagrue, E. Piette, and S. Tabary, "Constraint-based symmetry detection in general game playing," in *Proceedings of IJCAI'17*, 2017, pp. 280–287.

[19] C. Browne, "Deductive search for logic puzzles," in *IEEE Conference on CIG*, 2013, pp. 359–366.