ICCTA 2021

2021 7th International Conference on Computer Technology Applications

July 13-15, 2021 | Vienna, Austria | Virtual Conference

Association for Computing Machinery

ICPS Published by ACM

FH | JOANNEUM University of Applied Sciences

Association for Computing Machinery

On Using Monte-Carlo Tree Search to Solve Puzzles

Mohammad Sina Kiarostami, Mohammadreza Daneshvaramoli, Saleh Khalaj Monfared, Aku Visuri, Helia Karisani, Simo Hosio, Hamed Khashehchi, Ehsan Futuhi, Dara Rahmati, Saeid Gorgin.

Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu, Finland.
School of Computer Sciences, Institute for Research in Fundamental Sciences(IPM), Iran.

2021 7th International Conference on Computer Technology Applications (ICCTA 2021, former ICCIT), July 2021, Austria.

Contents

Introduction

- Solving puzzles are critical in the field of AI since they contains many general problems such as Pathfinding.
- Monte-Carlo Tree Search (MCTS) has surged into popularity as a promising approach due to its low run-time and memory complexity. --> We need to know how to employ MCTS.
- We study the applicability of MCTS in solving puzzles or solving a puzzle with MCTS and propose a general classification of puzzles (**four classes**) based on their features.
- We apply several optimizations to improve MCTS performance in terms of Time and Memory consumption.

Implementing Constraint-based MCTS for Puzzle Solving

- Puzzle Classification:
  - Class A
  - Class B
  - Class AB
  - Class C

Table 1. Classification of common logical puzzles based on our proposed *Classes*.

Puzzle	Class A	Class B	Class AB	Class C
Sudoku	X			
Kakuro	X			
Tetris				X
Clickomania		X		
Flood-it		X		
NumberLink	X	X	X	
Slitherlink	X	X	X	
Nonogram	X	X	X	
n-Puzzle		X		
Sokoban		X		
2048				X
Light-up	X			

Puzzle Classification

- Class A:
  - The puzzles in Class A are defined by the 3-tuple G=$\langle X,D,C\rangle$. The tuple X=($X1$,$X2$, ...,XN) represents the Variables (the unknown values to be found in the puzzle). The Domain of each variable is defined by the tuple D=($D1$,$D2$, ..., DN). The Constraints are defined by set of L constraints of C={$C1$,$C2$, ...,CL}, where each constraint is defined by Ci=(fi,di). Boolean function fi() is a primitive function and di with the size of mi, identifies the domain of Variables for fi().
  - The puzzle is considered to be solved if: $\bigwedge_{i=1}^{L} f_i(S_i) = True$
  - In this class, puzzles are *Time Independent*.

Puzzle Classification

- Class B:
  - The Class B of puzzles is directly affected by the step moves (time step), such as moving or drawing a line in a grid.
  - For instance: *Flood it*.
  - The puzzle is considered to be solved if:

$$\exists\, t_{goal} \in \{0, 1, 2, .., t_{final}\} : \forall t_j \leq t_{goal}$$

$$\forall i \in \{1, 2, ..., L\}, \forall k \in \{1, 2, ..., P\} :$$

$$u_i = \{\alpha_1, \alpha_2, ..., \alpha_{|u_i|}\}, d_k^{t_j} = \{\beta_1^{t_j}, \beta_2^{t_j}, ..., \beta_{m_k}^{t_j}\}$$

$$S_i = \{X_{\alpha_1}^{t_{goal}}, X_{\alpha_2}^{t_{goal}}, ..., X_{\alpha_{|u_i|}}^{t_{goal}}\}$$

$$O_k^{t_j} = \{X_{\beta_j}^{t_j}, X_{\beta_2}^{t_j}, ..., X_{\beta_{m_k}}^{t_j}\}$$

$$(\bigwedge_{t_j=1}^{t_{goal}} a^{t_j} \wedge \bigwedge_{j=0}^{t_{goal}} (\bigwedge_{k=1}^{P} f_k^{t_j}(O_k^{t_j})) \wedge \bigwedge_{i=1}^{L} g_i(S_i)) = True$$

Puzzle Classification

- Class AB:
  - This class is not a complete separated class. It means that the puzzles in this class can be defined and solved by Class A or Class B, as the problem solver prefers.
  - Both formulations of Class A and B could be applied.

Puzzle Classification

- Class C:
  - Other puzzles which are not classified in Class A,Class B, and Class AB, are left to Class C. These puzzles are often more complex and involve random input variable at each time-states.
  - Simply, we can not solve or play these puzzles on a paper.
  - For instance: *2048 puzzle.*
  - **This class of puzzles should not be solved with MCTS due to their at least one random input.**

Optimizations and Modifications to MCTS

- Order of Solving:
  - Reduces the branching factor by choosing a generic approach that lets the algorithm decide to start from where to solve the puzzle.
- Reliable MCTS:
  - If any non-terminal node returns a low value, the algorithm will not expand the node.
- Fast Rollout Function:
  - A general state is created in the first node's simulation, and all the following simulations only apply their changes.
- Accurate Randomness:
  - Instead of performing a complete random simulation, we consider some heuristics and validate them after each random move to direct the simulation without adding a considerable computation.

Results and Evaluation

Target Examples (Puzzles):

- Class A: Sudoku, Calcudoku, Light-up.
- Class B: Clickomania.
- Class AB: Slitherlink.
- Class C: *NA*

Results and Evaluation

- Class A: Sudoku, Calcudoku, Light-up
  - MCTS-Optimized is far better than others.

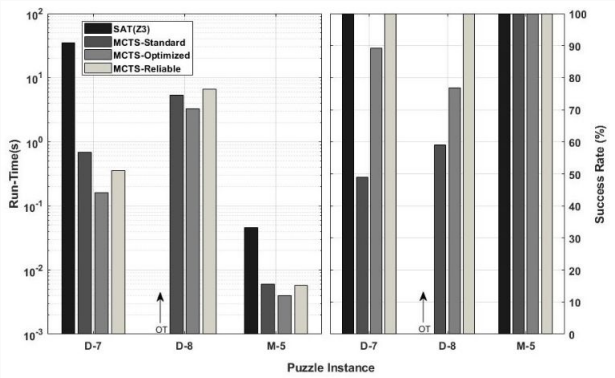

Fig. 3. *Calcudoku* Run-time and Accuracy Evaluation.

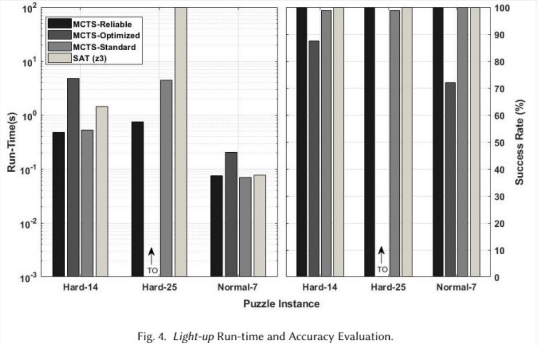

Fig. 4. *Light-up* Run-time and Accuracy Evaluation.

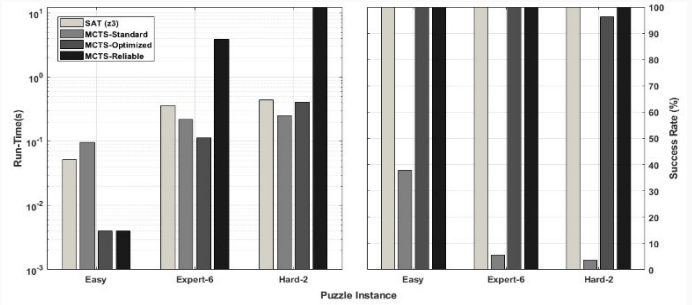

Fig. 2. *Sudoku* Evaluation. (Time and Accuracy of four different approaches are compared)

Results and Evaluation

- Class B: Clickomania
  - SAT solutions fail as an efficient approach mainly due to the wide search space caused by the *time* parameter.
  - MCTS-Optimized is better than other MCTS versions.

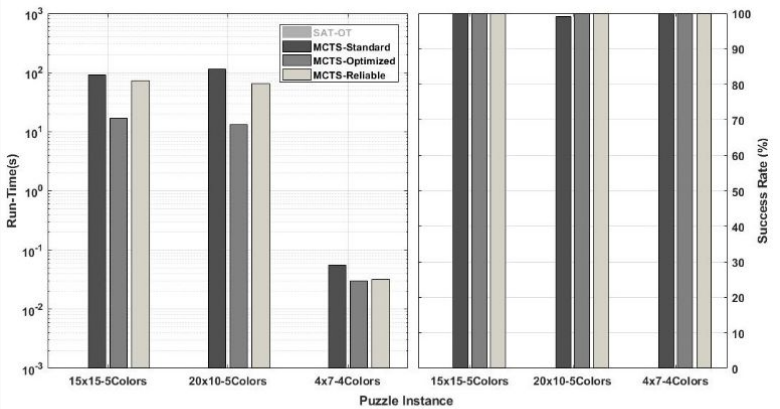

Fig. 5. *Clickomania* Run-time and Accuracy Evaluation.

Results and Evaluation

- Class AB: Slitherlink
  - In this example, we solve the puzzle based on Class B since the number of total variables of the such problem will be much higher in Class A representation. MCTS-Optimized outperforms other approaches.

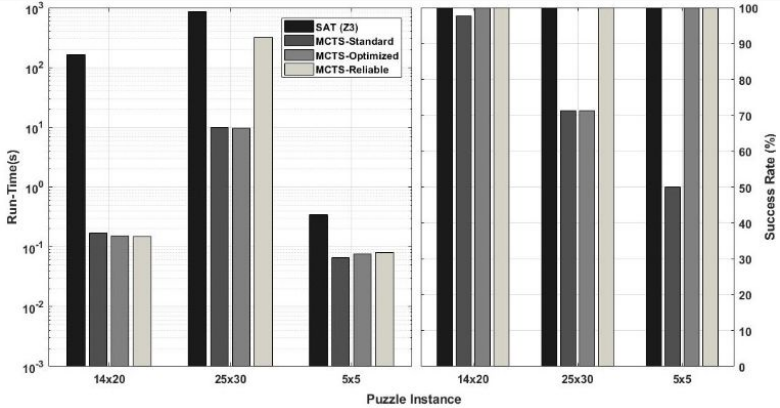

Fig. 6. *Slitherlink* Run-time and Accuracy Evaluation.

Conclusion

- We studied the applicability of the MCTS approach to solve logical single-player puzzles.
- We developed an infrastructural mathematical categorization that could be used to describe puzzles and suitably to be approached by MCTS.
- We indicated MCTS performs well both in accuracy and run-time in most logical puzzles compared to the conventional state-of-the-art SAT solvers.

Q&A

Thank you!

Mohammad Sina Kiarostami

Mohammad.Kiarostami@oulu.fi

Center for Ubiquitous Computing, Faculty of ITEE, University of Oulu, Finland