Stochastic Tree Search for Highly Coordinated Planning

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Bilal Kartal

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Prof. Stephen J. Guy and Prof. Maria Gini

August, 2016

Acknowledgements

I would like to take this opportunity to thank everybody who supported me to earn this PhD degree at a worldwide well-respected research university.

First and foremost, I would like to thank my advisor Stephen J. Guy. He encouraged me to perform research on challenging and interesting problems and gave me enough freedom to explore. He has introduced me to the field of stochastic planning three years ago which has evolved to my PhD thesis. Several times, Stephen sat down with me and help debugging my code. I am very lucky to have an advisor so accessible and helpful enabling me to progress for my thesis work quickly. Secondly, I would like to thank my co-advisor Maria Gini for her support and advices over the years. She always helped me to converge my research goals around an overarching theme. I am indebted to both Stephen and Maria for their countless hours of help for me to grasp both technical capabilities and responsibilities, more importantly the philosophical aspects of a PhD degree.

I would like to thank all the faculty members at the University of Minnesota that served on my committees during my PhD. I would like to thank Prof. Jarvis Haupt, Prof. Stergios Roumeliotis, and Prof. John Weissman for serving on my written and oral preliminary exam committees. I would like to thank my advisors, Prof. Victoria Interrante, and Prof. Jarvis Haupt for serving on my thesis committee.

I would like to thank many friends with whom I wrote research papers, had conversations about random and non-random topics, brainstormed for research ideas, and most importantly survived the stress of PhD. The list of people include everybody in Stephen's Applied Motion Laboratory and Maria's Artificial Intelligence, Robotics, and Vision Laboratory. Particularly I want to thank Julio, Vikas, Ernesto, Akash, Nick, Balu, and Baris for all the lunches and social events.

Lastly, but most importantly, I would like to thank my wife, my princess, Selda for her patience and never-ending support. She could earn herself a PhD degree with the effort she spent on keeping me on the track for this PhD achievement. We are thankful to have two lovely kids, our daughter Azra and our son Omer Alparslan who brought so much joy to our lives.

Dedication

Bu doktora tezi ülkem Türkiye Cumhuriyetine adanmıştır. (This PhD thesis is dedicated to my country, Republic of Turkey.)

Abstract

Coordination plays a key role in efficiently solving multi-agent problems such as exploration of unknown environments, search and rescue, and surveillance. Planning in a highly coordinated fashion causes traditional search algorithms to perform poorly as it creates combinatorial search spaces that renders exploration versus exploitation dilemma challenging as well. Recently, there has been great improvement in stochastic planning in large search spaces with sampling based algorithms. One particular algorithm is Monte Carlo Tree Search (MCTS) which is an anytime stochastic planning algorithm. MCTS employs the well-established multi-armed bandit theory to solve optimization problems by asymmetrically sampling actions where it foresees the existence of an optimal solution.

In this thesis, we propose new algorithms and heuristics in order to address several challenges arising due to highly coordinated planning in physical and abstract domains. Our algorithms improve scalability for search in large domains, provides data-driven evaluation functions to guide the search algorithm better, and enables finite search algorithms to produce infinite length solutions. In the first part of this thesis, we study two multi-robot planning problems that require high degree of coordination, patrolling and task allocation. The main challenge for these domains is the large search space. For patrolling, we propose a novel search technique, the Monte Carlo Tree Search with Useful Cycles, which can generate optimal cyclic patrol policies with theoretical convergence guarantees. For task allocation, we develop a parallelized MCTS based planner where branch and bound paradigm is integrated to the search algorithm for admissible pruning. In the second part of this thesis, we study two coordinated abstract planning problems within the field of procedural content generation, goal-driven computer narrative generation and Sokoban puzzle level creation. In these problems, virtual agents coordinate their actions to procedurally create stories and puzzles which have numerous application areas ranging from video games to education. We formulate both story generation and Sokoban puzzle generation as optimization problems and propose data-driven heuristic evaluation metrics for efficient coordinated solutions using MCTS.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many real-world problems have such structures, e.g. firefighting, logistics, and border patrolling where decision makers do not act independently, but they need to consider every possible decision the others can make in order to find efficient solutions. The coordination aspect of these problems create very large search spaces rendering traditional search algorithms ineffective. Planning algorithms need to efficiently localize important parts of this large search space to find reasonable solutions quickly. This process yields another challenge, i.e. the exploration versus exploitation dilemma where the planner needs to make decisions given uncertain quality of actions at every branching level. Recently, there has been great improvement in large-scale planning with sampling based algorithms where the quality of actions is measured by sampling towards future states. One example is Monte Carlo Tree Search [1, 2], an anytime stochastic planning algorithm which has been shown to perform well for large planning domains.

MCTS employs the theoretically well-grounded machine learning approach of Upper Confidence Bounds to addressing the exploration versus exploitation dilemma. Even though MCTS has been very successful in the game of Go [2] and several other domains [1], there are several challenges in applying it to highly coordinated physical and abstract planning domains. For example, in some planning domains, solutions are in the form of policies that are infinite length such as multi-robot patrolling and surveillance. However, existing search algorithms do not support infinite policy generation from finite search. Another challenge is that many planning domains such as those in procedural content generation has diverse action sets where finding well-suited evaluation functions is non-trivial. In this thesis, we present our contributions addressing these challenges in different application areas within the field of multi-robot planning and procedural content generation by adapting and extending sampling based stochastic search techniques.

In the first part of this thesis, we propose search based approaches for two $\mathcal{NP}$-hard multi-robot coordination problems, i.e. the multi-robot patrolling problem and multi-robot task allocation problem with time windows and capacity constraints. The patrolling problem investigates how to generate continuous patrol regions strategically to prevent intrusions and it has been an active research problem within the last decade, especially as more and more autonomous robots are available for surveillance tasks at much lower costs. Our approach generates cyclic continuous patrolling strategies optimally on perimeter patrolling, and near-optimally on arbitrary environments. We deploy our patrol algorithms on real robots and perform indoor experiments. A more general multi-robot planning problem that we study is the multi-robot the multi-robot task allocation problem with time windows and capacity constraints. This problem has many application areas such as surveillance and logistics where robot team must be allocated to tasks with spatial and temporal constraints. Our search-based approach, which also use branch and bound paradigm, quickly generates high quality solutions on complex problem scenarios and generalizes well over varying set of scenarios.

In the second part of this thesis, we apply MCTS to two problems within the field of procedural content generation, i.e. computer narrative generation and Sokoban puzzle generation. Narrative generation problem investigates how to automatically create stories in a coherent fashion. Application areas of computerized narrative generation are numerous such as automating text scripts in computer games, generating military training scenarios, possible uses in education, and summary generation for task-executing robots. For narrative generation, we propose an MCTS based planner along with new heuristics by formulating the multi-agent narrative generation problem as an optimization problem. Our approach aims to find plausible story events and create story plans accomplishing predefined goals where story domains are hand-crafted. However, one major bottleneck for the computerized narrative generation community is the manual authoring needed to define story domains consisting of actors, items, places, and the believability of certain actors performing certain actions. To address this challenge, we also propose a data-driven domain inference technique that can infer these domain parameters by employing semantic knowledge networks and existing story books. Many video games have puzzles either at their core or as a mini-game. Automatically generating these puzzles can improve game design phase, and keep games varied and more exciting. Secondly, we adapt MCTS for puzzle generation in a novel way where the puzzles are generated through simulated game play. This novel approach guarantees solvability in all generated puzzles.

1.1 Thesis Statement

My thesis will show that sampling-based stochastic tree search techniques can be adapted to efficiently find approximate solutions to highly coordinated problems that arise in a variety of domains. The resulting approaches are scalable, support data-driven evaluation functions, and can generate infinite length policies as needed for applications such as multi-robot planning and procedural content generation. These stochastic approaches can be successfully applied to domains with large state spaces, and their sampling-based nature allows flexibility in the details of the problem formulation and supports quick feedback for user-in-the-loop interaction.

1.2 Main Contributions

Coordination is important for planning problems as it yields efficient results. In this dissertation, we study planning problems in physical and abstract domains as shown in Figures 1.1 and 1.2. Here we briefly describe the characteristics of these problems.

- **Multi-robot patrolling:** For this problem, we seek to find *continuous* patrol policies for the robots to cover the patrolling areas or perimeters. However, since patrolling problem requires infinitely long patrol policies, standard search algorithms including MCTS is not suitable. We propose a stochastic search technique which augments the search with cyclic nodes that compactly represent continuous policies without breaking the convergence properties of MCTS. Robots must coordinate their actions to minimize the risk of intrusion. For state representation,

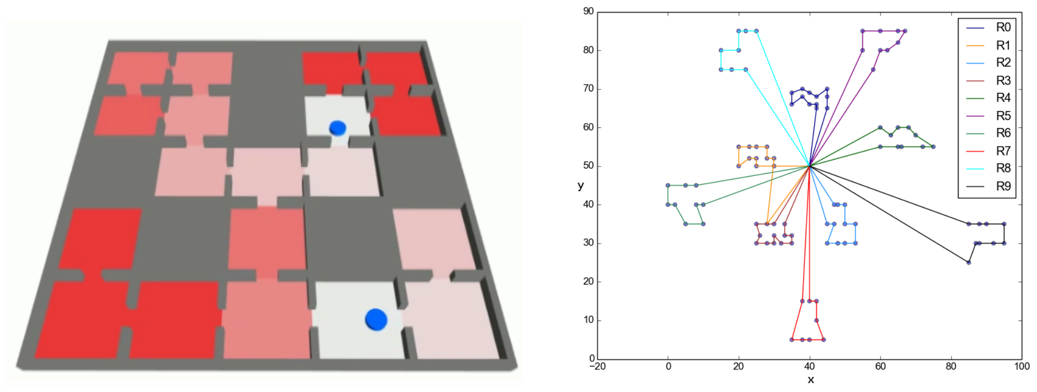
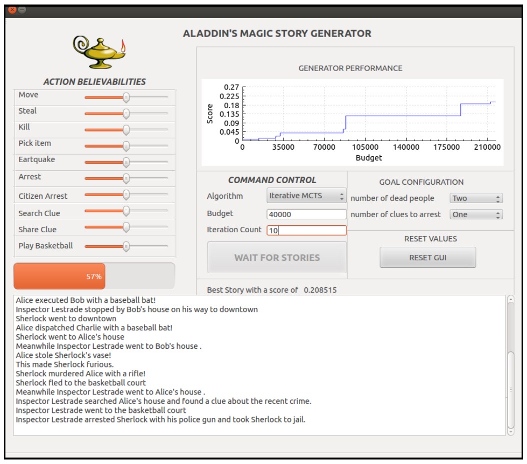


Figure 1.1: Overview of my contributions for coordinated planning in physical domains: **(Left)** The robots employ an optimal patrol policy obtained with our MCTS-UC approach. **(Right)** A near-optimal task allocation policy found by our parallelized approach with an approximation rate of 1.03 to an optimal solution is shown.

each tree node keeps robots' locations, and each transitioning action corresponds to robots moving to adjacent vertices on a graph environment. We will discuss this contribution in Chapter 3. Portions of this work have been published in [3].

- **Multi-robot task allocation:** For this problem, each robot must be allocated to a disjoint subset of tasks where each task has spatial and temporal constraints along with capacity and duration constraints. In a highly coordinated fashion, robots must share the tasks, and the overall distance cost of the robot team must be minimized. Tree structure is identical to the aforementioned one for patrolling. We will discuss this contribution in Chapter 4. Portions of this work have been published in [4, 5].

- **Multi-agent story generation:** In this problem, we firstly formalize narrative generation problem as a goal-driven multi-agent optimization problem where agents coordinate their actions to generate a story that satisfies the predefined story goals. Given that application areas include games and virtual reality, this problem necessitates efficient solutions with lower run-times which allows the approach to generate a variety of content quickly. We adapt MCTS so that the user-in-the-loop can interact with the planner by changing our proposed data-driven believability heuristic without modifying the MCTS planner. We will discuss this



Algorithm: Iterative MCTS
Budget: 40000
Iteration Count: 10
number of dead people: Two
number of clues to arrest: One

Best Story with a score of 0.208515

```
Alice executed Bob with a baseball bat!
Inspector Lestrade stopped by Bob's house on his way to downtown
Sherlock went to downtown
Alice dispatched Charlie with a baseball bat!
Sherlock went to Alice's house
Meanwhile Inspector Lestrade went to Bob's house .
Alice stole Sherlock's vase!
This made Sherlock furious.
Sherlock murdered Alice with a rifle!
Sherlock fled to the basketball court
Meanwhile Inspector Lestrade went to Alice's house .
Inspector Lestrade searched Alice's house and found a clue about the recent crime.
Inspector Lestrade went to the basketball court
Inspector Lestrade arrested Sherlock with his police gun and took Sherlock to jail.
```

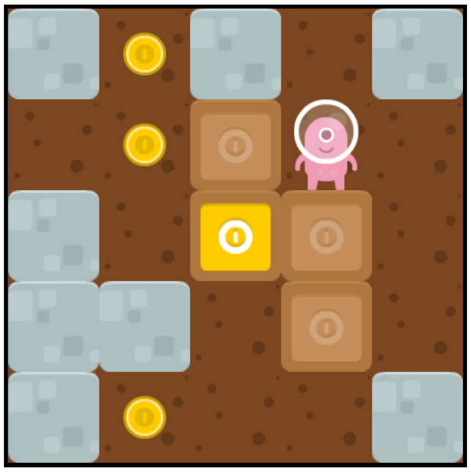


Figure 1.2: Overview of my contributions for coordinated planning in abstract domains: **(Left)** The GUI for our story generator is shown which enables the user-in-the-loop to alter story domain parameters interactively to interact with the planner. **(Right)** A high scoring 5x5 Sokoban puzzle generated by our method is shown where the goal is to move the agent to push boxes (brown squares) so that all goals (yellow discs) are covered by the boxes. Yellow filled boxes represent covered goals.

contribution in Chapter 5. Portions of this work have been published in [6, 7].

- **Sokoban puzzle generation:** In this problem, we propose a Sokoban puzzle generator with a novel concept that guarantees solvability in all puzzles without the computationally expensive step of solving the puzzle. In our planner two agents coordinate their actions to generate a non-trivial puzzle where the first agent initialize the puzzle while the second one intelligently play it to finalize the board. We perform user-studies and infer puzzle difficulty metrics with a data-driven approach to evaluate puzzles. Our approach proposes a search tree structure that compactly combines simulated game play into level generation search. Since MCTS is anytime, we can save intermediate puzzles which lets us save tens of puzzles in a single run. We will discuss this contribution in Chapter 6. Portions of this work have been published in [8, 9].

Chapter 2

Background

In this section, we present background information on the problems that are studied in this dissertation. As MCTS algorithm is at the core of my thesis, we will conclude this section with a detailed description of it.

2.1 Coordinated Multi-Agent Planning

Coordination can be defined as a common protocol that intelligent agents follow by considering each other's actions while decision-making and acting. Coordination plays a crucial role in effectively solving problems involving multiple agents, i.e. people and/or robots, where tasks can be divided into sub-tasks and shared among the agents such as pick-up and delivery problems. There are also scenarios where an individual task requires multiple agents simultaneously such as multi-robot box pushing.

Coordinated multi-agent planning is inherently a difficult problem due to the combinatorial explosion of possible actions [10, 11, 12]. For example, let's consider a simple planning setting where two agents Alice and Bob might move around different places. Their joint-space actions will create an exponential search space. At any decision level, joint space representation will include all possible permutations of places that Alice and Bob can go. The problem is further amplified by the number of decision levels if a specific path is to be generated for Alice and Bob while they arrive their goals. Agents coordinate their behavior to accomplish a goal, and depending on the problem, some optimization function must be developed which will direct the agents to accomplish their goal. We study different variants of coordinated multi-robot planning problems, i.e. multi-robot patrolling, and multi-robot task allocation problems. These problems have many application areas such as surveillance and logistics. Although explicit coordination requirement for these problems make them computationally harder due to large state spaces, sparse sampling based stochastic search techniques such as Monte Carlo Tree Search (MCTS) have been shown to perform well in problems with similar search spaces. Most notably, an MCTS-based computer Go player outperformed top human professional players on a 9x9 board [13] and very recently on a 19x19 full board [2] where Go game is currently one of the most challenging problems due to enormous branching factors AI community is working on.

2.2 Exploration versus Exploitation Dilemma in Planning

Multi-armed bandit problem [14] investigates how a gambler can maximize his cumulative rewards from a row of slot machines where each slot machine has a fixed but unknown reward distributions. The gambler has to find an ordering of machines to play to maximize his gain. However, each time a slot is used, the reward is drawn from its initially unknown distribution. This is a challenging task since the gambler both needs to learn the reward distribution and maximize his gain during the learning phase. Each time the gambler chooses a slot machine (trying an action in our setting), he records the reward and he can maintain statistics, e.g. average reward and how many times the slot machines is tried, about the slot machine to estimate its reward distribution. Exploration vs. exploitation dilemma is the tradeoff between choosing the best looking slot machine (exploiting) and trying sub-optimal looking (might be an optimal one) or new slot machines (exploring). Bandit theory has been employed in many domains such as signal jamming [15], computer vision [16], and medical applications [17]. Bandit theory provides a theoretical foundation for the exploration vs. exploitation dilemma. One of the machine learning techniques to address this dilemma is upper confidence bounds (UCB1) [18] as shown in Eqn. 2.1.

When multi-robot planing and procedural content generation problems are formulated as tree search problems, each level of the tree corresponds to a multi-armed bandit

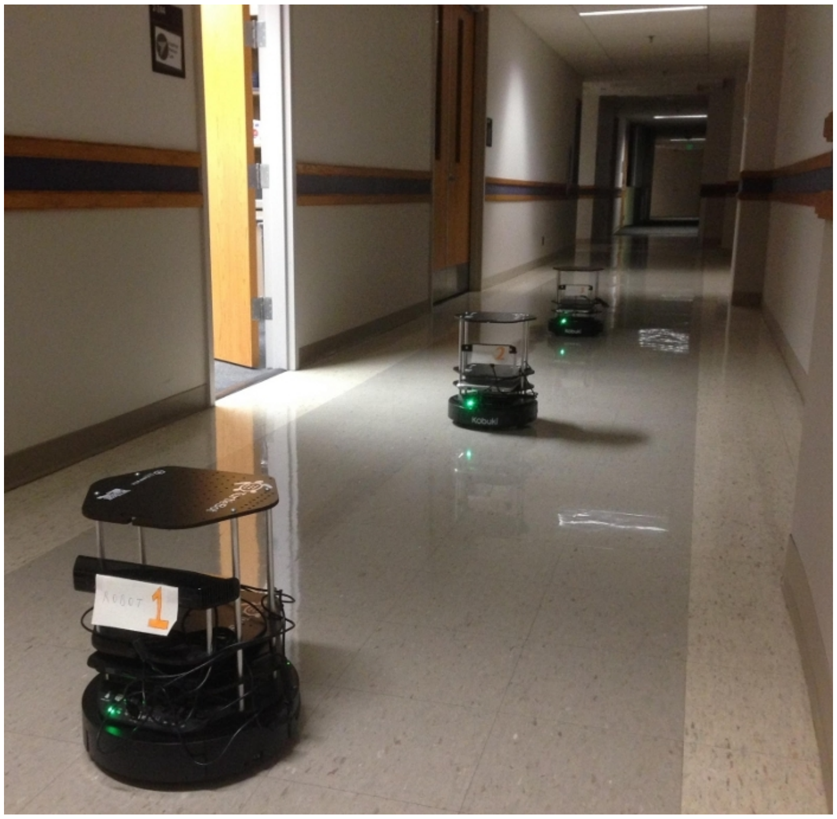


Figure 2.1: Turtlebot robots are in a patrol mission along the corridor following an optimal solution generated by our MCTS-UC approach. The robots are programmed by using ROS (Robotic Operating System) framework. We firstly map the corridor by creating a 2-D occupancy grid by using a particle filter based SLAM approach. Using this map, we developed a mission controller for patrolling that is built on top of ROS navigation and localization stacks.

problem where the exploration versus exploitation dilemma arises from the possible actions in the planning domain. UCB1 guarantees convergence to an optimal solution for finite horizon planning problems given infinite computation power and memory. Employing UCB1 method for searching tree structures is referred to as UCT (Upper Confidence bounds applied to Trees) [19] which also refers to MCTS indeed. In UCT approach, we need to devise an evaluation function in order to measure the quality of actions based on the problem domain. The evaluation function must be in the range of 0 and 1 for asymptotic convergence guarantees.

$$f(c) = W(\pi_c) + C\sqrt{\frac{\ln n_v}{c_v}} \tag{2.1}$$

where $W(.)$ denotes the average evaluation score obtained and let π keeps the policy (an ordered set of coordinated actions from the initial state), π_c is the parent's policy so far including node c, n_v is the total number of times the parent node n has been visited, and c_v is the total number of times that the action transitioning node n to node c has been previously tried. C is the constant that determines the exploration versus exploration tradeoff. The value of C must be $\sqrt{2}$ for theoretical convergence guarantees, but it can be tuned depending on the problem scenario.

2.3 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a stochastic search method that has been remarkably successful for large-scale planning problems. The key mechanism for MCTS is that it expands the most promising areas in the large search space with uneven tree growth. MCTS has been applied for a variety of challenging planning problems. Monte Carlo Tree Search is a best first search algorithm that has gained traction after its breakthrough performance in the game Go on a smaller game board [13]. Indeed, very recently an MCTS based distributed approach remarkably beat a top human professional on a full size Go board [2]. Other than game AI [20, 21, 22, 23, 24, 25, 26, 27, 28], MCTS has been employed for a variety of domains such as planning for Physical-TSP problem [29], single vehicle transportation planning [30], computer vision [31, 32], game level generation [33] and large POMDPs [34]. A recent work experimented different strategies for final strategy construction is presented in [35]. The authors [36] analyzed different types of evaluation functions for MCTS and show that the success of the algorithm depends on the smoothness of the domain. The authors in [37] proposed a method to improve performance by better accounting for transpositions. We refer the reader to the excellent survey on MCTS for further information [1].

Overall MCTS algorithm consists of four main operations, i.e. selection, expansion, rollout, and back-propagation as shown in Alg. 1 where ω represents full strategy including random rollout actions and $R(\omega)$ represents the evaluation score, i.e. quality, of the full strategy. In this algorithm, budget is the number of simulations which needs to be set to lower values for interactive applications, and but can be set higher given memory and time constraints. We present MCTS steps in in Figure 2.2 by describing each step in the algorithm in detail. After back-propagation steps, the selection step is started from the root node again. This way, the tree can grow in an uneven manner, biased towards good policies. After a fixed number iterations, we construct the final strategy by walking down the tree from the root node by recursively selecting action sets with highest visit counts until hitting a non-parent node which is observed to perform more robust. For completely deterministic problems, we can also keep the best policy found so far after rollouts, and use the best one after search is finished or halted.

MCTS is an anytime algorithm, i.e. it returns the best found solution if the search algorithm is halted at any time. As MCTS is a sampling based technique, the algorithm must be given the environment dynamics and the protocols defining how the agents will interact with both environment and each other. The root of the tree keeps the state information about the agents and their defined properties. The edges correspond to transitioning actions which can modify both agent and environment properties, and each node keeps updated states.

Algorithm 1: MCTS Algorithm

```
Input  : Budget
Output: Policy
while $Budget > 0$ do
    Node ← ucbSelection(root) ;
    $\omega$ ← rollout(node);
    backpropagate($R(\omega)$) ;
    Budget = Budget−1 ;
end
```

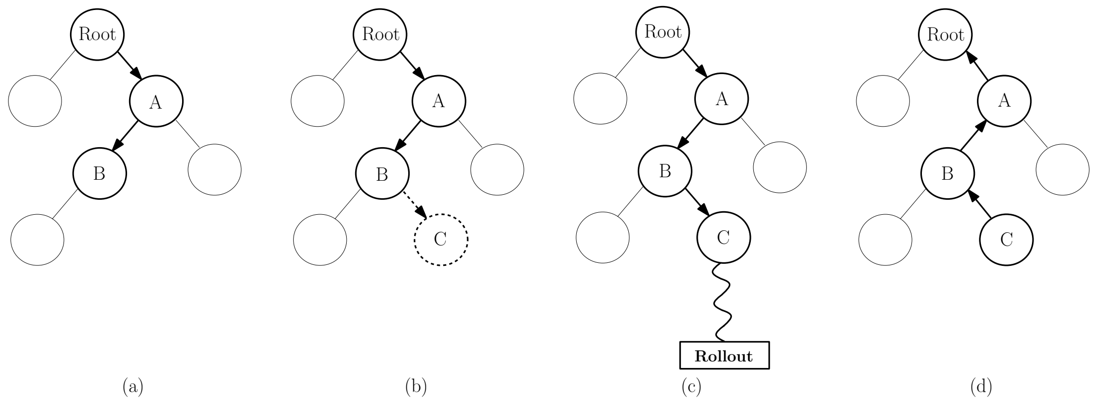


Figure 2.2: **Overview of Monte Carlo Tree Search algorithm:**
(a) **Selection step:** Starting from the node, UCB1 equation is employed recursively until a node with explored action is selected. As shown in Figure 2.2a, nodes A and B are selected.
(b) **Expansion step:** A new node corresponding to a unexplored state is added to the search tree as shown in Figure 2.2b, node C is added to the tree.
(c) **Random Rollout:** A series of random actions are taken from the expanded node, i.e. node C, to complete the deterministic but partial strategy as actions corresponding to the nodes A,B, and C are unlikely to be a complete strategy.
(d) **Back-propagation step:** Full strategy consisting of selected tree node actions and random rollout is evaluated and the score is back-propagated from the expanded node to the root, i.e. nodes C, B, A, and the root.

Chapter 3

Multi-Robot Patrolling

3.1 Introduction

Robots are nowadays commonly used to perform critical tasks, such as search and rescue operations [38], intelligent farming [39], mine sweeping and environmental monitoring [40, 41, 42]. The robots employ sensing capabilities for both localization and monitoring purposes [43]. In such tasks, robots have to observe or sweep an environment by solving a coverage planning problem [44, 45, 46]. A variant of this problem is the multi-robot patrolling problem, in which multiple robots must coordinate their motions in order to minimize the probability of intrusion. Unlike coverage, patrolling needs to be performed continuously. Efficient patrolling methods can be used in many settings such as country borders and smaller areas in cities to increase the safety of citizens, or making sure that everywhere in large environments is sweeped for cleaning purposes [47].

The multi-robot patrolling problem has been an active research area within the last decade, especially as more and more autonomous robots are available for surveillance tasks at low costs. This problem is $\mathcal{NP}$-hard in its general sense [48]. Therefore, several heuristics and approximation algorithms have been propose, with efficient solutions found for simple cases. We adapt Monte Carlo Tree Search (MCTS) algorithm [19] to generate patrolling policies across arbitrary environments. MCTS can successfully search in large domains by using random sampling. The algorithm is anytime and converges to optimal solutions given enough time and memory for finite-horizon problems.

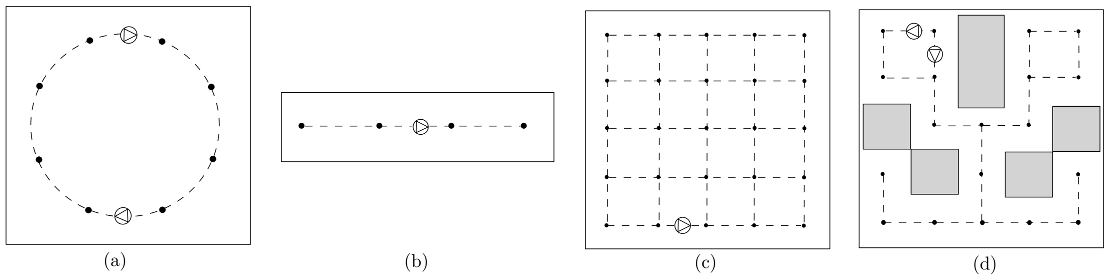


Figure 3.1: Examples of patrol graphs used in our simulation experiments for n robots:(a) A circular scenario where $|V| = 8$ and $n = 2$. (b) A line scenario where $|V| = 4$ and $n = 1$. (c) A grid scenario where $|V| = 25$ and $n = 1$. (d) A grid scenario with obstacles where $|V| = 19$ and $n = 2$.

3.2 Main Results

One of the main challenges to adapt MCTS to the patrolling domain is the ability to generate infinite length policies; the policies generated by MCTS are valid for a small time horizon while patrolling task has to be performed continuously. We address this issue by introducing Monte Carlo Tree Search with Useful Cycles, MCTS-UC, which augments standard MCTS with *cyclic nodes* to return infinite, cyclic policies.

We firstly propose the use of stochastic tree search for patrolling policy generation. Second, we show how useful cycles can be incorporated into MCTS to efficiently generate continuous cyclic policies without losing convergence guarantees. Finally, we experimentally show the applicability of MCTS-UC across a variety of scenarios. Coupled with a pruning heuristic, our approach can generate policies for intractably large environments.

3.3 Related Work

Multi-robot patrolling problem is an optimization based problem where multiple robots must cover and patrol an environment continuously in a coordinated fashion to prevent intrusions. It is a variant of coverage problems [47, 49, 50], and it has been studied since at least the work of [51] and a diversity of patrolling strategies was theoretically analyzed in the following years [48, 52, 53]. Patrolling strategies for a team of robots can be computed either in a centralized manner where a central entity computes the policy for all the robots [54], or in a decentralized manner [55]. In general, centralized approaches lead to more optimal policies than its decentralized counterparts, but are more computationally demanding as the policy space is exponential with respect to the number of robots in the environment. We plan over the joint patroller space and convert the resulting centralized policy into individual policies that each of the robots should follow.

A diversity of strategies for patrollers have been propose to account for static and stochastic intruders [56, 57, 58], including multiple intruders performing coordinated attacks [59]. Decentralized approaches are present in [60, 61, 62], and a temporal-logic based approach for persistent surveillance is present in [63]. Previous work has also focused on extending the longevity of the patrolling task by replacing robots based on their battery life [64]. Patrolling problem has also been extensively studied in a game theoretic context where interactions between patrollers and intruders are modeled as a leader-follower game [54, 65]. In a recent work [66], a trust model has been propose such that poorly performing patrollers are identified and patrolling tasks are reassigned dynamically. The authors in [67] propose an approach minimizing the communication latency for the patrolling task. We refer the reader to the survey in [68] for more information about the patrolling problem.

3.4 Problem Formulation

In our problem setting, we are given n patroller robots, $r = \{r_1, \ldots, r_n\}$, that have to periodically cover an environment to guard it from intrusions. For simplicity, we model the environment as an undirected graph $G = (V, E)$, where the vertices V denote the patrol regions and the edges E represent the connectivity between these regions as shown in Figure 3.1. We assume that time can be discretized, and elimination of an intruder is instantaneous. Initially, at $t = 0$, each robot is placed at vertex 0. At each discrete time step, the possible actions for a robot are to move to a neighbor vertex in G or to stay still. Multiple robots are allowed to occupy the same vertex simultaneously.

We assume that an intruder q enters the environment at a specified time t_e, and it takes t_p time steps to complete a successful attack. As typically assumed in the literature (e.g. [57]), the intruder has the same motion model as the patroller robots, but this can

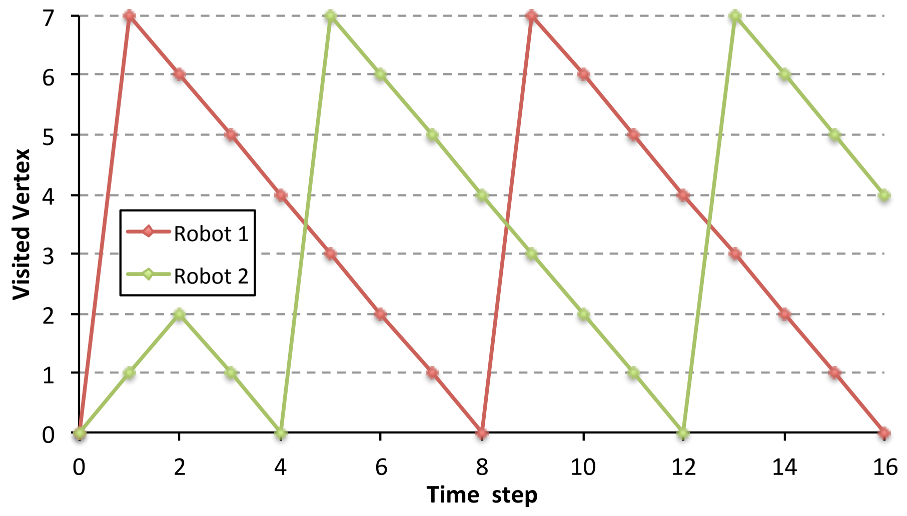


Figure 3.2: Trajectories of 2 robots on a perimeter, $|V| = 8$. Both robots start at the same vertex and they adjust their placements to equidistant vertices and patrol in the same direction continuously. V_7 and V_0 are adjacent vertices.

also be altered to different models easily. We further assume that the intruder can enter the environment from any vertex in V. Two intruder models are considered:

1. Dynamic: The intruder enters at a random vertex at the same time as the patroller robots, $t_e = 0$, and performs a random walk exploring the environment.

2. Stationary: The intruder enters at a random time $t_e \geq 0$, at a random vertex and spends t_p time-steps at the location completing the intrusion.

Given a patroller team r and a single intruder q following one of the above models, our goal is to find a *joint* policy π for the robots that maximizes the likelihood of capturing the intruder before the attack is successful. We evaluate any policy π using the following function:

$$R(\pi) = \begin{cases} 1 & \text{if } \exists r_i \in r \text{ s.t. } r_i(t) = q(t) \text{ and } t \leq t_e + t_p \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where $r_i(t)$ denotes the location of r_i at time t and $q(t)$ is the intruder's location at t.

From Eq. 3.1, we assume a robot can only sense an intruder on the vertices of G

at discrete time steps. We assumed a discrete sensing model since continuous sensing causes more power usage and noisier observations that might be caused by robot motor noise or camera image blur.

Importantly, this assumption makes the problem more challenging in the presence of dynamic intruders; if the intruder moves past a patroller while it crosses an edge, the intruder will not be detected.

3.5 Policy Generation

In this section, we explain how the MCTS approach can be employed to the multi-robot patrolling problem in order to generate near-optimal finite length trajectories for the patrolling robots.

An overview of the MCTS is present in Algorithm 1. The algorithm maintains a tree structure where each node represents the complete state of the world, s_t, consisting of the locations of the robots at a certain time t, with t also referring to the depth of the node in the tree. The root of the tree contains the initial state of the world that corresponds to the initial locations of robots. Each link in the tree represents one possible joint action set consisting of n actions, one per each robot. The MCTS algorithm proceeds by repeatedly adding one node at a time to the current tree until a given budget (e.g., number of simulations) is met. The newly added node represents the resulting world state after applying the corresponding joint action to the robot team. For each potential joint action set, we keep track of how many times we have tried it, and what its average evaluation score is.

MCTS generates policies through uniform random *rollouts*. A simulated policy ω consists of a sequence of joint action sets:

$$\omega = \{a_1, a_2, \dots a_z, \xi_1, \xi_2, \dots \xi_x\}, \tag{3.2}$$

where each a_i refers to deterministic action sets obtained from the existing tree and each ξ_i refers to uniform random action sets. The task of random rollouts is to provide a probabilistic evaluation of incomplete deterministic policies. After evaluating ω using Eq. 3.1, the resulting $R(\omega)$ is used to update the average evaluation scores from the leaf node to the root node while incrementing their visit counts, in a process known as *back-propagation.*

3.6 Monte Carlo Tree Search with Useful Cycles

Standard MCTS is limited to produce a finite length policy for finite search budgets as the search tree can only grow to a fixed depth. However, the patrolling problem requires the generation of infinite policies. To address this issue, we propose an MCTS variant, Monte Carlo Tree Search with Useful Cycles (MCTS-UC) that generates continuous cyclic policies for the patroller team. The notion of useful cycles has been previously studied in [69] to improve path quality on probabilistic roadmaps. Following their work, we define a useful cycle as a set of patrolling paths that starts and ends in the same vertex set for all robots. Therefore, we exploit the spatial similarity of visited vertices of patrollers, i.e. whether the same set of vertices are visited between any two states or not, to determine a useful cycle.

In terms of implementation, MCTS-UC creates artificial cyclic nodes which represent continuous policies. These nodes will be part of the tree search during exploration-exploitation. Our MCTS-UC algorithm is summarized in Algorithms 2-3.

Algorithm 2: Rollout with cylic action sets

```
Input: leafNode
if leafNode == CyclicArm then
    PerformCyclicActions() ;
else
    RandomRollout();
end
```

To find a useful cycle and generate a cyclic node, our approach first needs to determine the spatial similarity of robot locations among different states. We assume that two states are equivalent if the same set of vertices are occupied by the patrollers in both states. We check for equivalent states during back-propagation step (see Algorithm 3).

Consider, for example, two equivalent nodes A and B as shown in Fig. 3.3a. Given these equivalent nodes, a cyclic node, node C, is created as a sibling arm to node B and its cyclic parent node is set to node A as depicted in Fig. 3.3b. Node C will be part of UCB1 selection phase just as any ordinary arm. When node B is selected after

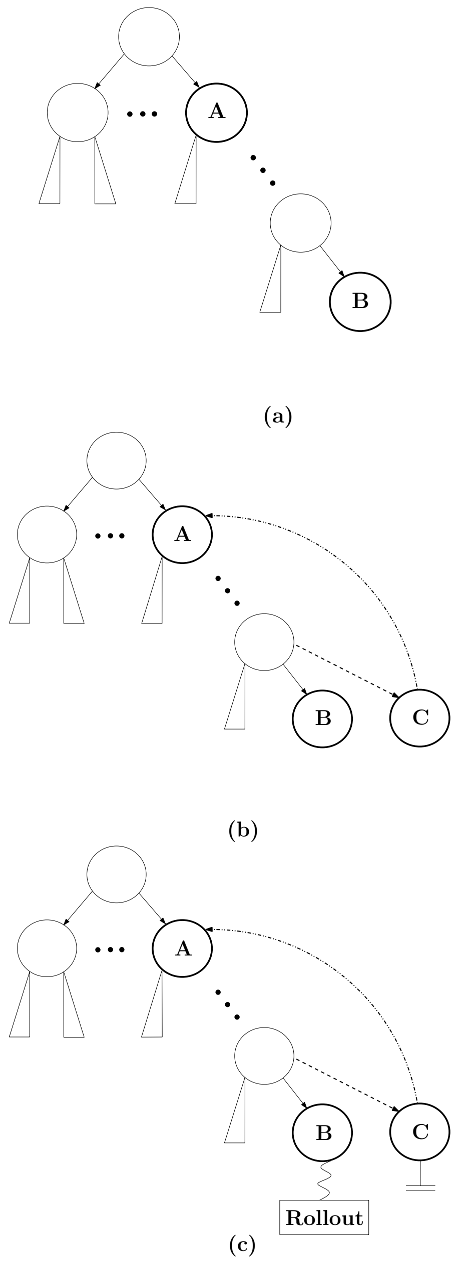


Figure 3.3: Overview of Monte Carlo Tree Search with Useful Cycles: (a) During the back-propagation of node B, node A is found to have the same state. (b) A new cyclic node C is created to capture the cycle. (c) While node B is evaluated with standard rollouts, node C is evaluated cyclically.

Algorithm 3: Back-propagation with cyclic arm creation

```
Input: leafNode
tempNode ← leafNode.getParent();
while tempNode != root do
    if tempNode == leafNode then
        leafNode.CreateCyclicSiblingArm();
    end
    tempNode ← tempNode.getParent();
    UpdateVisitCounts();
    UpdateWinrates();
end
```

creating node C, we do a roll-out and expand the tree as in standard MCTS. However, if the cyclic node C is selected in further iterations of MCTS-UC, our algorithm creates a cyclic action buffer by pushing actions one by one while walking up the tree from itself to its cyclic parent (i.e. node A) and continuously performs these actions (Fig. 3.3c), evaluates the cyclic policy and back-propagates the policy score through the *non-cyclic* parent nodes as in standard MCTS. We should also note that both nodes A and B storing the equivalent states are kept unchanged to maintain the integrity of MCTS convergence conditions.

Our approach, MCTS-UC, maintains convergence properties of MCTS for finite horizon problems as we can obtain a cyclic policy in a bounded patrol environment. We should note that the UCB1 equation (Eq. 2.1) guarantees that exploration always continues, and this leads to asymptotic convergence. However, we should state that as MCTS-UC has additional cyclic nodes at different levels of the tree, the required time to converge to the optimal policy might increase. For instance, if cyclic nodes close to the root node lead to sub-optimal but much better policies than that of sibling nodes, we might end up having longer convergence time.

3.7 Convergence of MCTS-UC

MCTS-UC shares the convergence properties of the original MCTS for finite horizon problems. On one hand, the UCB equation (Eq. 2.1) guarantees that exploration always continues. On the other hand, the artificially added cyclic arms' rewards are also identically and independently distributed by Hoeffdings inequality as cyclic nodes are also evaluated for the same probabilistic intruder model. Similar to MCTS, the convergence analysis of MCTS-UC is also based on non-stationary arms having rewards sequences satisfying two drift conditions: 1) The expected average value of the arms has to converge to their true values $\overline{X}_{in} = \frac{1}{n}\sum_{k=1}^{n} X_{ik}$, where X_{ik} refers to the reward of i'th arm and k'th trial. From our policy evaluation function (Eq. 3.1) we know that $0 \leq X_{ik} \leq 1$ holds. Since we employ cyclic action sets repeatedly when cyclic nodes are selected, cyclic arms converge to their true values faster, empirically, as cyclic action sets lead to the coverage of the same regions repeatedly. 2) The tail distribution criteria should also be satisfied [19]. In MCTS-US this is true, since all cyclic and non-cyclic arms' rewards are identically and independently distributed.

We should note that as MCTS-UC has additional cyclic nodes at different levels of the tree, the required time to converge to the optimal policy might increase. For instance, if cyclic nodes close to the root node lead to sub-optimal but much better policies than that of sibling nodes, the search algorithm might end up having longer convergence time.

3.8 Pursuit and Evasion Simulation Experiments

To demonstrate the suitability of standard MCTS in policy generation, we apply it to the pursuit-evasion problem described by Noori et al. [70]. Here the task is to find an optimal pursuing strategy on a line (see Fig. 3.1(b)) for a modified random walker evader that always moves either left or right, where $t_e = 0$ and t_p varies. The authors employed an MDP analysis to show that their results are near optimal. For this problem, MCTS generates policies with competitive capture probabilities for a budget of 1M as shown in Table 3.1. The resulting policies are approximately equal to the scores reported in the MDP analysis across all the authors' scenarios, slightly over-performing for smaller policy lengths and slightly under-performing for larger policy lengths. This near-optimality motivated our extensions to MCTS for the patrolling domain as explained in the following section.

	$t_p = 9$	$t_p = 14$	$t_p = 19$	$t_p = 24$	$t_p = 29$
Noori et al. [70]	0.297	0.429	0.564	0.710	0.803
MCTS	0.301	0.432	0.569	0.692	0.740

Table 3.1: Capture probabilities for a line where $|V| = 21$.

3.9 Patrolling Simulation Experiments

We evaluated our MCTS-UC algorithm by comparing it to standard MCTS in terms of capture probability in three different environments: a line, a perimeter, and a 2D grid as shown in Fig. 3.1. We considered the two types of intruder strategies described in Section 3.4.

As previously discussed, infinite length policies are required in the multi-robot patrolling domain. Unlike standard MCTS, our MCTS-UC approach can successfully generate such policies. Table 3.2 shows the capture probabilities of a stationary intruder on a perimeter (Fig. 3.1(a)), where the number of vertices $|V|$ is 15 and the number of patrollers n is 3. As it can be seen in Table 3.2, for large simulation budgets MCTS-UC leads to policies with higher capture probabilities. However, for smaller budgets up to 500K, MCTS is slightly better than MCTS-UC. The reason behind this is that MCTS-UC invests some of its exploration budget on creating cyclic nodes, which pays off when a larger budget is available ($\geq$ 3M simulations), converging to an optimal cyclic policy given a budget of 5M simulations. In contrast, MCTS produces short-term policies for the same budget sizes, and is intractable after 5M explored nodes.

In a grid environment with $n = 2$ and a dynamic intruder (Fig. 3.1(c)) MCTS-UC outperforms MCTS for varying t_p's as shown in Table 3.3. Note that the capture probability never reaches 1, as intruders that cross paths with patrollers (in edges) are not captured. In the same scenario with $n = 1$ against a stationary intruder, MCTS-UC creates policies with capture probability of 0.81, and 1 for $t_p = 25$, and $t_p = 50$ respectively.

Budget	1K	10K	500K	3M	5M	6M
MCTS	0.1778	0.231	0.2698	0.3012	0.3125	-
MCTS-UC	0.1612	0.171	0.255	0.3217	1	1

Table 3.2: Capture probabilities in a perimeter with $|V| = 15$ and $n = 3$.

	$t_p = 30$	$t_p = 40$	$t_p = 50$
MCTS	0.532	0.551	0.592
MCTS-UC	0.626	0.774	0.829

Table 3.3: Capture probabilities in a grid environment with $|V| = 25$ and $n = 2$. Budget is fixed to 1280K nodes.

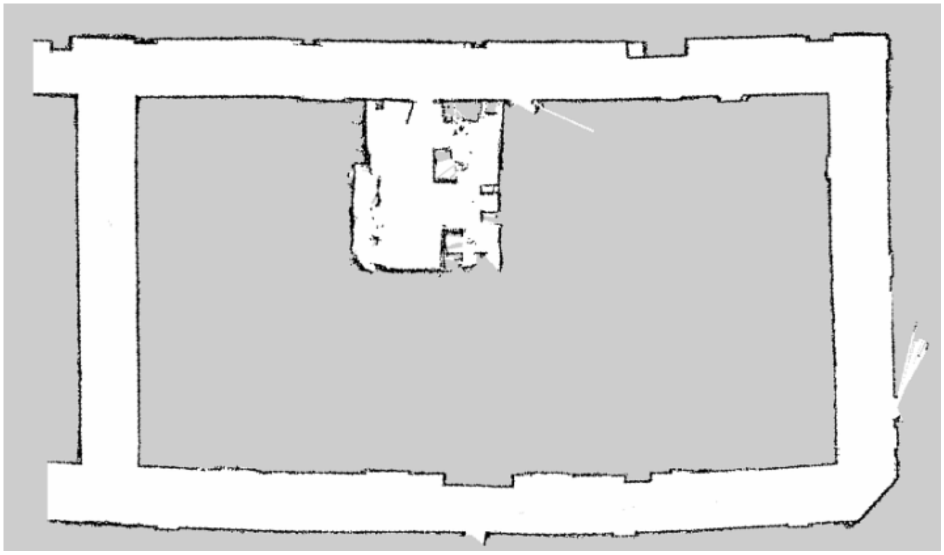


Figure 3.4: The created map of the corridor in Keller Hall 2nd floor at University of Minnesota is shown. We employed Gmapping ROS package based on a particle filter approach. The created map is represented as a 2D occupancy grid. Robots compute global plans based on this map for navigation while reacting to real-time collisions by still sensing with a local planner.

3.10 Real Robot Patrolling Experiments

We performed experiments with multiple Turtlebot [71] robots by developing a ROS (Robotic Operating System) [72] based controller for the patrolling mission. The experiments took place on the second floor of Keller Hall at University of Minnesota campus. We have three Turtlebot 2 robots that are equipped with Kinect like sensors as shown in Figure 2.1.

For autonomous robot navigation, the robots both keep track of their positions and orientations and learn the environment that they are navigation in. SLAM (Simultaneous Localization and Mapping) deals with this problem where robots need to solve two problems at the same time, i.e. learning the map of environment and localizing themselves while creating a map. There are a variety of SLAM algorithms in literature such as those based on Extended Kalman Filter [73] and Particle Filters [74, 75]. ROS provides several libraries for SLAM algorithms. The authors in [76] analyzed and compared these ROS based SLAM libraries extensively on different environments with different assumptions on the available sensor data. SLAM algorithm that use only camera sensor perform poorly on environments with long corridors due to high spatial similarities. For our setting where there are long corridors in the environments, gmapping which harnesses both camera sensor and odometry information performs better. We present the generated map in Figure 3.4 which is obtained by tele-operating a single robot with a joystick on corridors couple of times for loop-closure. The created map is employed by all robots.

3.10.1 Robot Experiments Results

We have performed two sets of patrolling experiments on the map shown in Figure A. In the first experiment, we generate a policy against a uniform probabilistic intrusion model. In the second experiment, we have a non-uniform probabilistic intrusion model where we weigh the lab much more than the corridors. In both settings, our approach quickly finds solutions with high capture rates.

Patrolling with Uniform Intrusion Model

For this experiment, we have employed a uniform intrusion model where it is equally likely that there will be an attack on any point on the map. Our algorithm has generated an optimal solution in less than two minutes where robots disperse in the environment and continously patrol in the same direction. We present robot trajectories in Figure 3.5. In this experiment, penetration time t_p is set to 8. The policy our algorithm has produced resulted in a capture rate of 99%. Some intrusions are still possible as it takes a while for robots to disperse in the environment to make their first visits to every location.

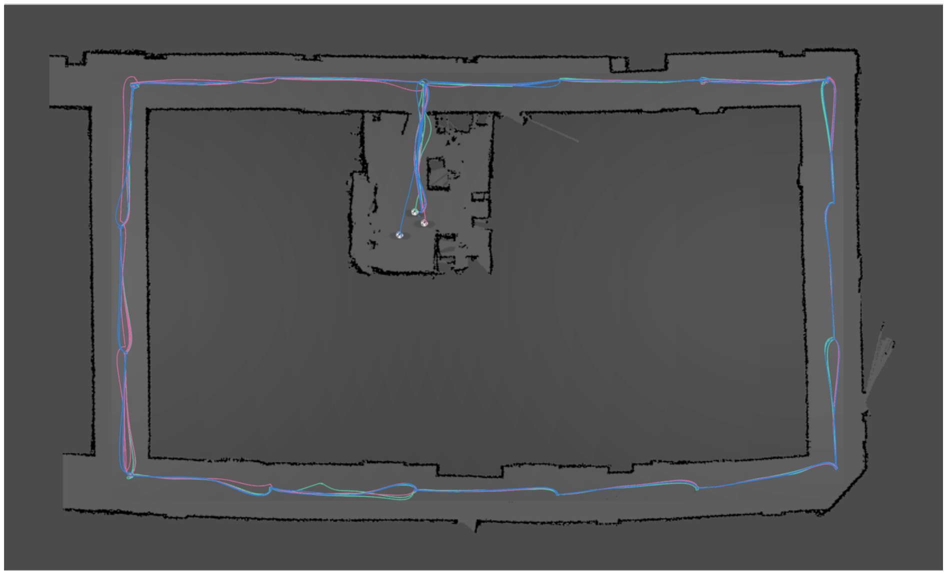


Figure 3.5: Robot trajectories are shown with different colors. The robots start from the the lab, i.e. the square region inwards from the corridors. We employed a uniform intrusion model for the whole map. All the robots cover the whole the map by dispersing to equidistant locations.

Patrolling with Non-Uniform Intrusion Model

In this experiment, we have employed a non-uniform intrusion model where the lab, i.e. the square region inwards from the corridors, has been assigned a higher intrusion probability than the corridors. Our method has found a solution in four minutes with a capture rate of 81% where $t_p = 15$ has been employed. Figure 3.6 shows the real robot trajectories following this policy where one robot covers the lab and hallway immediately outside (green path), while the others two robots cover larger sections of the corridor away from the lab (pink and blue paths). This experiment shows that our approach can be extended to different intrusion distribution models.

3.11 Performance Analysis

Scalability: Besides having better capture probabilities than MCTS, MCTS-UC uses an order of magnitude less memory than MCTS. This is because whenever a useful cyclic node is selected, the search tree will not grow for that simulation as the cyclic

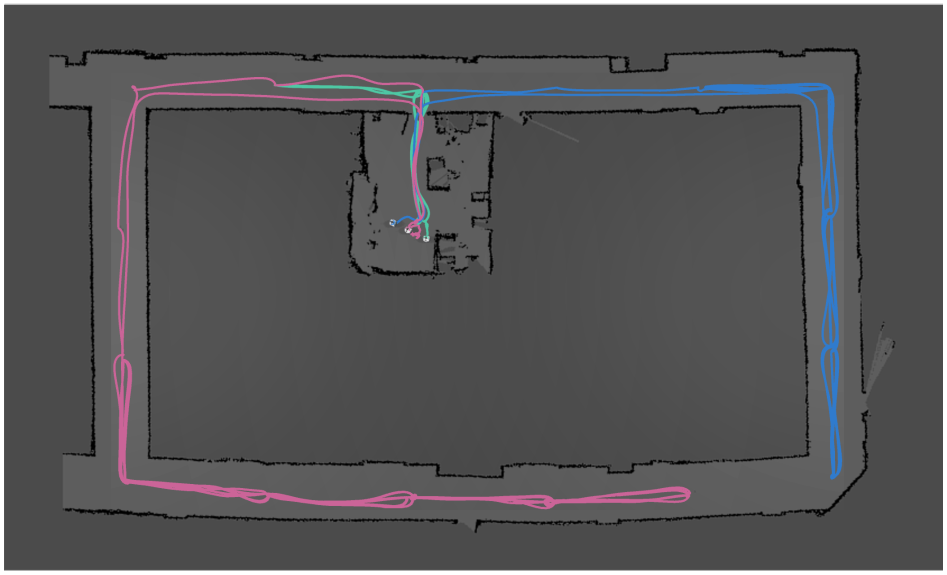


Figure 3.6: Robot trajectories are shown with different colors. The robots start from the the lab, i.e. the square region inwards from the corridors. We assigned a higher risk of intrusion for the lab and one robot (shown with green trajectory) mostly stayed in the lab while covering very small portion of the corridor from time to time.

action set will be performed repeatedly. However, computing policies for all robots can be prohibitively expensive for large environments. In these cases, heuristics that reduce the search space can mitigate the problem. For this purpose, we evaluated MCTS-UC with the Iterative heuristic on a 20x20 grid where $|V| = 400$, $n = 2$, and $t_p = 200$ for a dynamic intruder. We compare the capture probabilities of MCTS-UC with and without the Iterative heuristic. In this scenario, even a simple sweep of the environment requires 200 optimal moves which makes the problem intractable for MCTS-UC without the Iterative heuristic when using large budgets.

Runtime: The runtime of MCTS based approaches relies on the branching factor and the depth of the search tree. The Iterative heuristic provides runtime improvement in later simulations as it increases the number of initial committed steps. It can be seen in Fig. 3.7 that Iterative MCTS-UC is able to find better policies with larger budgets in less time. For the perimeter scenario shown in Fig. 3.1(a), MCTS-UC requires 10 seconds to obtain an optimal policy, and the grid scenario shown in Fig. 3.1(b) requires

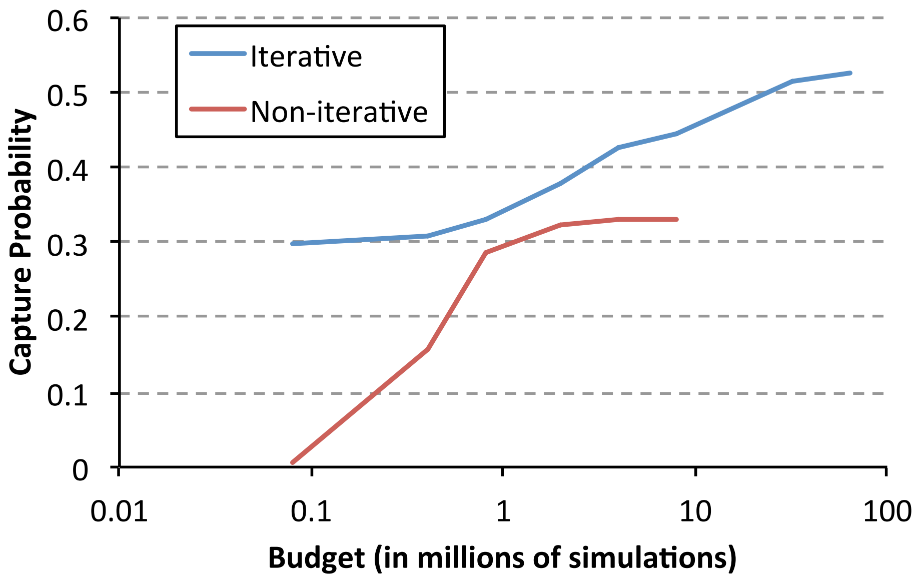


Figure 3.7: Capture probabilities of Iterative MCTS-UC as compared to standard MCTS-UC (x-axis is log-scaled).

one minute for an optimal solution with MCTS-UC.

3.12 Conclusions

We have present an anytime approach capable of generating continuous policies for patrolling problems with different intruder models. Our approach exploits spatial similarity of different robot configurations with time shifts to create infinitely long policies from finite simulation budget. It generates theoretically proven optimal policies for perimeters and near-optimal policies for arbitrary environments. One limitation of using MCTS-UC in our domain is the high branching factor due to joint-action space. While its spatial complexity scales polynomially with respect to the environment size, the degree of each node scales exponentially with respect to the number of robots, e.g. grid environments with $n \geq 5$ are intractable for MCTS-UC as the branching factor exceeds one thousand.

In next Chapter, we will present our contribution for multi-robot task allocation problem, i.e. a more generalized version of many planning problems where robots coordinate how to share tasks among each other to minimize overall team distance cost.

Chapter 4

Multi-Robot Task Allocation

4.1 Introduction

Multi robot systems generally necessitate a high level task allocation planner for efficiency. We study the multi-robot task allocation (MRTA) problem with temporal and capacity constraints with homogeneous robots (henceforth MRTA-TW). This problem falls under the category single-task robot, single-robot task, time extended allocation with in-schedule dependencies ID[ST-SR-TA] [77], although the constraints herein considered are far more complex than the ones specified in the taxonomy.

In MRTA-TW, a fixed set of tasks is allocated to a team of robots such that the total distance traveled by all robots is minimized, and temporal and other side constraints are satisfied. This problem is $\mathcal{NP}$-hard, even when a single-robot is considered. The single-robot version of the problem is a variant of the elementary shortest path problem with resource constraints, which is a well-known $\mathcal{NP}$-hard problem. For this reason, optimal methods are not practical for datasets involving tens of robots and hundreds of tasks due to large run times (several hours to days for hundreds of tasks and tens of robots). Hence, efficient but suboptimal solutions are needed.

Approximate centralized and decentralized methods have been propose for task allocation problems (e.g. [78, 79, 80, 81, 82, 83, 84]). While there exist some efficient schedulers (e.g. [80]) for problems in which distances are small enough to be subsumed into tasks' durations, little attention has been given to solvers for problems in which

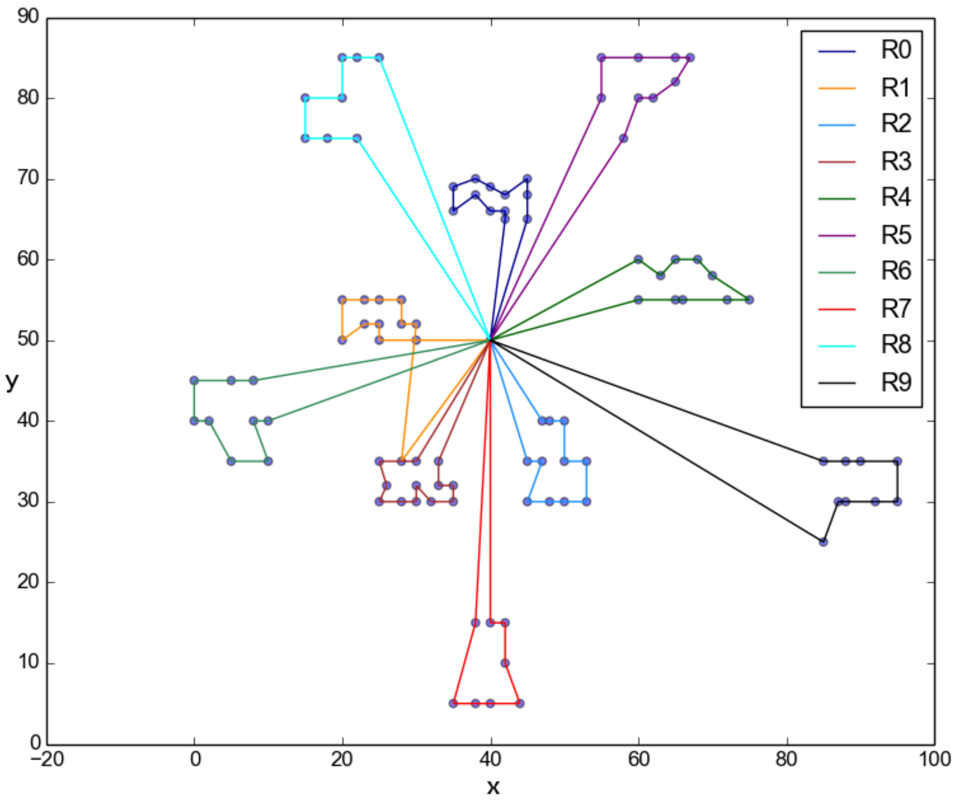


Figure 4.1: A near-optimal solution generated by MCTS for C101 test scenario in Solomon benchmark. This solution has an approximation rate of 1.03 to an optimal one. $R_i, i \in \{0, 1, 2, \ldots, 9\}$ indicate the robots.

routing and scheduling problems have to be solved simultaneously. Our approach tackles the latter problem and provides a solver that achieves asymptotic optimality, while attaining optimal or near-optimal results for non-trivial number of tasks and robots within practical run-times.

4.2 Main Results

This Chapter describes our MCTS-based planner for MRTA-TW problem. To propose an efficient planner that can well generalize over different problem scenarios, we improve upon MCTS in three fundamental ways: (1) We design a multi-objective evaluation function that balances between minimizing the distance traveled by all the robots and maximizing the task completion rate. (2) We propose a customized root parallelization scheme that helps the algorithm better explore the solution space. (3) We use branch and bound to prune parts of the search tree that do not improve upon the best found solution to improve speed and solution quality.

Our solution's quality is assessed empirically using the Solomon data set [85] for vehicle routing problems with time-windows (VRPTW), and compared with many optimal results from VRPTW. Our method finds solutions that are at most 1.59 away from optimal ones, and achieve an average of 98% task completion rate across all data sets.

4.3 Related Work

Our work draws knowledge from the multi-robot task allocation (MRTA) with time windows. There have been previous attempts to deal with variants of our problem in the MRTA literature. The authors in [86] studied the allocation of tasks that have to be done at a certain location and have constraints on their start time. They offer a Mixed Integer Linear Programming (MILP)-based solution and a more scalable tabu-search algorithm. Unlike our problem, they only consider precedence ordering of tasks, and not hard temporal constraints on tasks. Heuristics based approaches have been propose as well such as [87].

The authors in [88] propose a MILP-based heuristic for allocating tasks with deadline constraints to heterogeneous robots. They consider only problems where task types and robot capabilities have to match. Like in [86], their MILP-based approach does not scale to large number of tasks and robots.

Task allocation with resource contention has been recently studied in [89] as well. The closest work to ours is present in [90], who explore the vehicle routing problem with location choice. The main difference between the latter work and ours is that they include precedence constraints. Precedence constraints impose structure on the ordering of tasks. With pre-processing, the ordering would reduce the size of our search space.

Our problem can also be cast as a VRPTW [85]. There is a rich literature of

Algorithm 4: MCTS with Search Parallelization

```
Input  : Budget
Output: Policy
$D(\hat{\pi_{best}}) = \infty$ ;
ThreadId = get_thread_num();
bestLocalScore[ThreadId] = $\infty$ ;
while $timeElapsed \leq Budget$ do
    Expanded_Node ← ucbSelection(root[ThreadId]) ;
    $\hat{\pi}$ ← rollout(Expanded_Node) ;
    if $f(\hat{\pi}) >$ bestLocalScore[ThreadId] then
        bestLocalScore[ThreadId] = $f(\hat{\pi})$;
        bestLocalPolicy[ThreadId] = $\hat{\pi}$;
        if $m' = m$ AND $D(\hat{\pi}) < D(\hat{\pi_{best}})$ then
            set_lock(&writelock);
            if $D(\hat{\pi}) < D(\hat{\pi_{best}})$ then
                $D(\hat{\pi_{best}}) = D(\hat{\pi})$;
                $\hat{\pi_{best}} = \hat{\pi}$;
            end
            unset_lock(&writelock);
        end
    end
    backpropagate($f(\hat{\pi})$) ;
end
```

centralized methods for VRPTW [91, 92]. Most of the heuristic methods that yield good quality solutions efficiently require modeling and parameter tuning specific to a data set, which makes these methods hard to generalize. Instead, our approach, which is based on Monte Carlo Tree Search, is more general and is guaranteed, given enough time, to converge to an optimal solution.

4.4 Approach Overview

We formalize the MRTA-TW problem and describe how we adapt MCTS to address it. We introduce our propose evaluation function and present how branch and bound paradigm can be applied within the MCTS algorithm, along with a simple but novel search parallelization method.

4.4.1 Problem Formulation

In the MRTA-TW problem, there are m tasks which need to be allocated to n robots. The objective is to minimize the total distance that the robots must travel to reach the m tasks while satisfying the temporal and capacity constraints. We assume $n \leq m$. Each task has a x-y location, service time, capacity demand, and time window. Demand is the number of capacity units consumed. A time window specifies the temporal constraints for task execution, by specifying the earliest time to start the task, and the latest time to end it. Time windows can have different lengths and are allowed to overlap.

We assume point mass robots, each with an x-y coordinate, a capacity, and a global deadline by which it has to return to a designated depot. We assume that a task needs only one robot to be executed, and a robot can perform several tasks, one at a time. Hence, the ultimate goal is to compute a route for each robot that starts and ends at the depot. A robot assigned to a route should be able to execute each task within the task's time window, and have enough time to travel between tasks and return to the depot. Each task should be included in only one route, and routes should cover all the tasks.

The problem is modeled as a directed graph, $G = (V, E)$, where the vertices V are the locations of the tasks and the depot. The set of weighted edges E includes only feasible edges obtained from pairing vertices in V. Edge $e_a^b \in E$ is a feasible edge if and only if task b can be completed after task a without violating the time-window constraints. The weight on the edges represent the distance between the pair of tasks. In our MCTS model routes are turned into allocation policies.

Let $\pi_i = \{\{r_i, t_i^1\}, \{r_i, t_i^2\}, ..., \{r_i, t_z^{|\pi_i|}\}\}$ denote the *individual task allocation policy* of robot r_i, where t_i^j corresponds to the j-th allocated task in the policy of robot r_i and t_z corresponds to the depot. For each employed robot, the *return to depot* action is included as a dummy task. Let $\hat{\pi} = \{\pi_1 \cup \pi_2 \cup .... \cup \pi_n\}$ denote the *global task allocation policy* for the entire set of robots, where each task is allocated to a single robot. In addition, let $D(\hat{\pi})$ represent the total distance traveled by the robots while following the policy $\hat{\pi}$. A complete policy is one where all tasks are allocated, i.e., $|\hat{\pi}| = m + n'$, while in an incomplete policy some tasks remain unallocated, i.e., $|\hat{\pi}| < m + n'$ where $n' \leq n$ is the number of robots that performed at least one task. Our objective is to find a *complete* policy $\hat{\pi}$ such that $D(\hat{\pi})$ is *minimized.* Once the search is over, the solution

$\hat{\pi}$ is decomposed into n individual robot policies to be executed simultaneously.

4.4.2 Approach Overview

We formulate the MRTA-TW problem as a tree search and propose an MCTS based method as outlined in Alg. 4. In our tree structure, robots are employed sequentially, i.e., the route for robot r_{i+1} will be computed once robot r_i returns to the depot.

At each level of the tree, a single robot is assigned one of the remaining tasks or returns to the depot. Once a robot returns to the depot it cannot be allocated any other task. This creates unbalanced allocations, however it keeps the branching factor at a manageable size.

During the search, the UCB algorithm is used to choose which task to allocate to each robot, as shown in Eqn. 2.1. Given that our problem formulation is completely deterministic, we can store the best found solution during the search and optionally halt the search with some solution quality threshold by exploiting the anytime property of MCTS.

4.4.3 Policy Evaluation Function

The MCTS algorithm needs an evaluation function to estimate the true rewards of tree actions by measuring the quality of full policies extended with random rollout actions. Evaluation functions are straightforward in most games where the player gets a payoff of 1, 0.5, and 0 for winning, tying, and losing, respectively. Evaluation is more complex for MRTA-TW because neither the value of the optimal solution is known nor most candidate solutions allocate all the tasks. The evaluation function in (6.3) helps find complete allocations that take distance and allocation percentage into account.

Let m' represent the number of allocated tasks for task allocation policy $\hat{\pi}$. Due to the nature of highly constrained properties of tasks, most candidate solutions fail to allocate all the tasks. However, MCTS still needs to direct the search to find better solutions. In this work, we propose an anytime policy evaluation function which guarantees that given more planning time, MCTS will find better solutions. Let α denote a loose upper bound on the total traveled distance $D(\hat{\pi})$, and $\hat{E}$ be a sorted version of E (in descending order of their distances). Then, $\alpha = 2 \times \sum_{i=1}^{m+n} e_i$ where $e_i \in \hat{E}$. Let

δ denote task completion of the candidate policy where $\delta = 1$ if $m' = m$, and $\delta = 0.5$ otherwise.

To discourage incomplete policies, we define a negative reward parameter, ψ, which is computed as follows: $\psi = 2 \times \sum_{i=1}^{m-m'} e_i$, $e_i \in E'$, where $E' \subset E$ is the set of edges directed from completed to uncompleted tasks, from uncompleted to uncompleted tasks, and from uncompleted tasks to the depot sorted in descending order of distances. While calculating ψ, we use the simple observation that no edge in graph G will be traveled twice.

Based on these definitions, we propose an evaluation function $f(\hat{\pi})$, to assess policy $\hat{\pi}$, as follows:

$$f(\hat{\pi}) = \frac{\alpha - (D(\hat{\pi}) + \psi)}{\alpha} \times \delta \tag{4.1}$$

Our evaluation function considers the worst case insertion cost of the unfulfilled tasks from the remaining feasible edges for penalty term. This helps MCTS to avoid getting trapped by partial policies with small total distances which are unlikely to complete all tasks.

Our evaluation function guarantees that $f(\hat{\pi_a}) > f(\hat{\pi_b})$ holds if $\hat{\pi_a}$ completes all the tasks and $\hat{\pi_b}$ misses at least one task. To show this, we can simply show bounds of $f(\hat{\pi_a})$ and $f(\hat{\pi_b})$ separately, which would imply our claim.

Case 1: $0.5 \leq f(\hat{\pi_a}) < 1$. We can observe that $D(\hat{\pi}) \leq \alpha/2$ is true as α is accumulated from the longest $m + n$ edges on graph G and multiplied by 2, while $|\hat{\pi}| \leq m + n$ depending on the number of robots utilized. Therefore, $\psi = 0$ and $\delta = 1$ holds by completion of all the tasks which completes our claim for case 1.

Case 2: $0 < f(\hat{\pi_b}) < 0.5$. This holds true because by missing at least one task δ will be set to 0.5, and $D(\hat{\pi}) + \psi > 0$ holds for any policy. The δ parameter behaves like a step function.

Lastly, our evaluation function returns monotonically increasing values along any path from the tree root as more tasks are allocated. For every allocated task, the actual traveled distance increases by some $c \geq 0$, while ψ decreases by at least c; this guarantees that the evaluation score of the policy never decreases for any sequence of actions in the tree as more tasks are completed.

4.5 Application of Branch and Bound

Branch and bound is used to prune nodes during search, based on the incumbent solution that allocates all the tasks. A simple observation is that no matter how many robots are utilized, each edge in E will be traveled at most once. We let $D(\pi_{\hat{best}})$ denote the total distance traveled corresponding to the best solution found so far that completes all the tasks.

Given a partial task allocation policy π_p which completes m_p tasks using n_p robots with a total distance of $D(\hat{\pi_p})$, there are two cases to consider. First, let's assume that all remaining robots, $n - n_p$, are at the depot. Then we generate a possible edge list, $E_{rest} \subset E$ by considering all edges from the depot to the uncompleted tasks, from uncompleted to uncompleted tasks, and from uncompleted tasks to the depot. Second, if there is a robot that completed a task but did not return to depot yet, we also consider edges from the robot location to uncompleted tasks and to depot for E_{rest}. In case there is no robot at depot, we neglect edges directed from the depot. During bounding tree branches, if there are fewer robots than tasks, we assume that each robot has to complete at least one task while a lower-bound on the future distance to travel is computed along tree branches.

We branch new nodes if the following condition holds $D(\pi_{\hat{best}}) - D(\hat{\pi_p}) > \sum_{i=1}^{q} e_i$ where $e_i \in E_{rest}$ and E_{rest} is sorted in ascending order. Here, q is the minimum number of edges to cover to complete a partial allocation policy, which is computed as follows; if $n < m$, then $q = (m + n) - |\pi_p|$, otherwise $q = m - m_p$.

4.6 Parameterized Root Parallelization for MCTS

Several different search parallelization methods have been propose for MCTS, such as tree parallelization, leaf parallelization and root (or single run) parallelization, as summarized in [1]. Among these different approaches, root parallelization has been shown to perform best for the game Go [93]. The authors in [94] show the effectiveness of root parallelization by exhaustive experiments over different problems. Root parallelization simply creates multiple search trees, one per thread, and merges the search trees once the search budget is complete to generate policies. It has minimal overhead as the threads do not communicate until the merging step.

We propose a novel variant of root parallelization, henceforth *parameterized root parallelization*. Each tree is given a different UCB exploration parameter so that they can explore the search space in different ways. Similar to pure root parallelization, our approach also creates multiple independent search trees per thread. There is a globally shared variable, $D(\pi_{\hat{best}})$, keeping the best complete solution distance found by any tree. The design idea for our approach is that finding a locally optimal solution early can better calibrate the search direction for all threads given the very large state space. This also improves memory efficiency as we can prune existing nodes that are guaranteed to not beat the best found so far.

For parallelization with k cores, we can set UCB exploration parameters as follows; $\hat{C} = \{\frac{2C}{k}, \frac{4C}{k}, \ldots, C, 2C, 3C \ldots, (\frac{k}{2}+1)C\}$ where $C = \sqrt{2}$. The first half of the cores are assigned smaller exploration parameters so that they can search deeper early to find a complete solution to expedite pruning in all trees, while the second half of the cores will explore more so as not to get trapped by a local maximum. In cases where none of the threads is able to find a task allocation policy which completes all the tasks, the result of our parallelization approach is identical to running MCTS k times with different C values with a single core and returning the best found solution consecutively.

4.7 Experimental Setup and Results

We assessed our MCTS method using the Solomon dataset for vehicle routing on commodity hardware. We used 8 logical cores on an Intel Core i7-4790 3.6 GHz Quad-Core computer with 32GB RAM. The algorithm is run once for each data instance with search times set to 1 hour. We present a near-optimal solution generated by our approach in Figure 4.1. We show how branch and bound improves our results for all categories in Figure 4.2. Overall anytime behavior of our approach is present in Figure 4.3.

The Solomon dataset provides a rich variety of problem scenarios for task locations and time windows. Tasks are either clustered (C), randomly scattered (R), or a mix of clustered and randomly scattered (RC). Each of these categories has either tight time windows (type 1) or large time windows (type 2). The individual spatiotemporal categories (e.g. C1, R2) have between 8-12 individual instances.

Each instance has 100 tasks, each task has a uniformly generated demand drawn

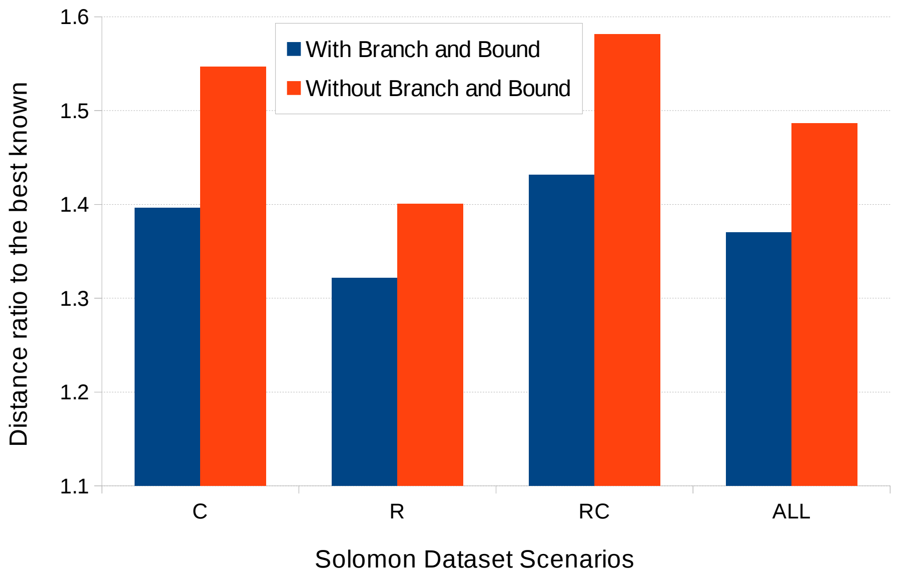


Figure 4.2: Smaller is better. Embedding branch and bound in the MCTS algorithm improves the overall distance ratio to the best known solutions 11% across all instances in the Solomon data set.

from $\mathcal{U}(1, 50)$, and service times of either 10 (C and RC data instances) or 90 (R data instances). The earliest start time for the time windows is drawn from $\mathcal{U}(1, 1000)$ and $\mathcal{U}(1, 3400)$ for types 1 and 2 data instances, respectively. The latest finishing times are drawn from $\mathcal{U}(20, 1000)$. The x-y coordinates of the tasks and robots are uniformly drawn from $\mathcal{U}(0, 100)$. The global deadlines for the robots have values between ~ 200 and ~ 3500.

4.7.1 Comparison to other methods

As no single method produces the best results for all problem instances in Solomon benchmark, we compare our results with the best known results [95] obtained from up to 16 methods including metaheuristics, local search, ant-based, and genetic algorithms. While these methods are more efficient, unlike our method, they neither generalize well across data sets nor do they provide any guarantees. We summarize our results in Table 4.1 and Table 4.2.

In Figure 4.2, we present how branch and bound technique is improving our results for all categories of the benchmark. Overall quality increase with branch and bound is about 11%, i.e. computed by considering all completed instances in the benchmark as branch and bound approach is activated only when a complete solution is found. Exploring the search space by eliminating branches which have no chance of improving the current best complete solution proves to be useful as it enables the search algorithm to explore the rest of the search space with more budget.

We report results for two scenarios: when the number of robots is fixed from the start, and when this number is as large as the number of available tasks.

4.7.2 Fixed number of robots

In this set of experiments the number robots used is upper bounded by the number of robots used in the best known solutions from the VRPTW literature.

Task allocation percentage results for type 1 and 2 data instances are present in Figure 4.3. Our algorithm allocates more than 40% of the tasks after 5 minutes; it allocates nearly all tasks across all datasets within one hour. The average task allocation rate and the percentage of instances in which all tasks are allocated are reported in Table 4.2. The algorithm yields high average allocation rates across all datasets, and it attains relatively lower percentages of instances in which 100% of tasks are allocated.

We observe that our approach in general is more successful for test instances where the number of robots is large and time windows are not very large. For example, the algorithm struggles to fully allocate tasks in C2 test instances, where the number of robots is 3 for all instances, while it does better in all type 1 instances where there are 9 or more robots. In type 2 instances each robot performs many more tasks. Having more robots enable random rollouts to complete more tasks. As shown in Figure 4.3, in 5 minutes we obtain much higher completion rates in type 1 scenarios. Also, we obtain best solutions for RC2 in type 2 instances as shown in Table 4.2. Our observation is also supported by the fact that in average RC2 has at least 50% more robots than other type 2 scenarios.

Detailed distance ratio results are reported in Table 4.1 and Table 4.2. The results reported in Table 4.1 show that our algorithm yields solutions with quality at most 1.59 away from the best-known for completed instances. These results are corroborated by the average distance ratio results in Table 4.2.

Similar to the completion results, our algorithm yields larger distance ratio values

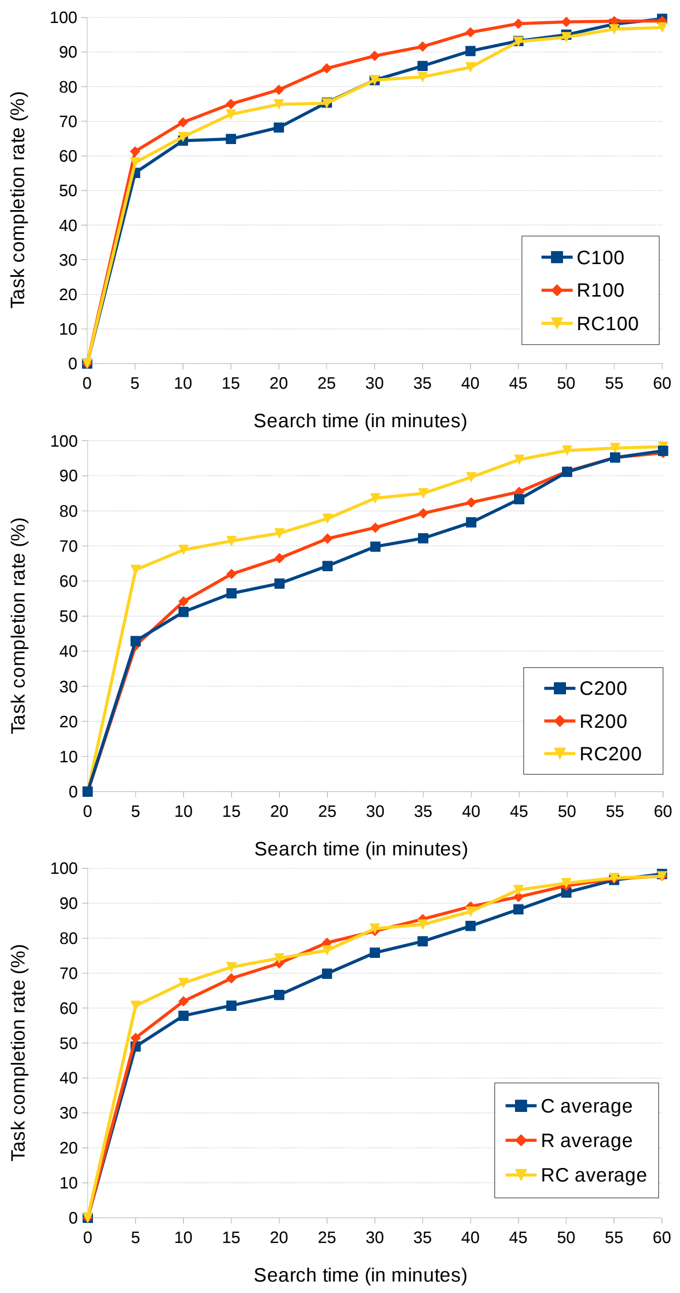


Figure 4.3: Task completion rates versus search time with MCTS for problems from the Solomon dataset. On average, MCTS achieves 50% task completion rate in 5 minutes with the help of random rollout extensions for allocation policies. With more planning time, we achieve up to 98% completion rate over all categories.

Scenario	MCTS	Optimal	Task %	Ratio to Opt.
C101	853.5	827.3	1	1.03
C102	1287.7	827.3	1	1.55
C103	1320.5	826.3	1	1.59
C104	1249.9	822.9	1	1.51
C105	1038.2	827.3	1	1.25
C106	982.6	827.3	0.99	-
C107	1117	827.3	0.98	-
C108	1076.1	827.3	0.99	-
C109	1173.9	827.3	1	1.42
R101	1820.8	1637.7	1	1.11
R102	1716.5	1466.6	1	1.17
R103	1593.4	1208.7	1	1.31
R104	1303.1	971.5	1	1.34
R105	1640.9	1355.3	1	1.21
R106	1532.9	1234.6	1	1.24
R107	1363.9	1064.6	0.99	-
R108	1051	960.9	0.96	-
R109	1428.6	1146.9	1	1.25
R110	1381.8	1068	1	1.29
R111	1436.5	1048.7	1	1.37
R112	1055.2	982.1	0.93	-
RC101	1515.2	1619.8	0.96	-
RC102	1851.4	1457.4	1	1.27
RC103	1554.2	1258	0.98	-
RC104	1373.7	1135.5	0.94	-
RC105	1973	1513.7	1	1.30
RC106	1520.8	1424.7	0.91	-
RC107	1614.1	1207.8	0.99	-
RC108	1515.2	1114.2	0.99	-

Table 4.1: Comparison of our MCTS solutions to the best known solutions (found by 16 different methods) on Solomon Benchmark

	C1	R1	RC1	C2	R2	RC2	Avg.
Completion Rate Avg.	0.99	0.99	0.97	0.97	0.97	0.98	**0.98**
% Instances all completed	0.67	0.75	0.25	0.00	0.27	0.63	**0.43**
Ratio to Opt. Distance	1.39	1.25	1.28	-	1.51	1.48	**1.38**

Table 4.2: Summary of results for MCTS using the same number of robots of the best known solutions for Solomon benchmark.

in type 2 data instances compared to type 1 ones. As stated before, the larger time windows cause the algorithm to evaluate more policies. Hence, we argue that larger run times would improve solution quality, given that the algorithm would be able to eventually focus the search away from allocations with larger distances.

4.7.3 Free number of robots

Given that it is challenging to complete all the tasks through random sampling with the tight robot team sizes we obtained from the best known solutions, we have experimented our approach by keeping everything the same but only changing the number of robots to be the same as the number of tasks to guarantee task completion with random rollouts. This setup also simplifies our evaluation function as $\psi = 0$ and $k = 1$ both hold. However, as expected, using more robots causes extra distance cost of leaving the depot and coming back.

Scenarios	C	R	RC	**All**
Distance Ratio	2.01	1.41	1.58	**1.67**
Team-size Ratio	2.65	2.11	1.92	**2.23**

Table 4.3: The ratio of found solutions to the best known are present for MCTS with free number of robots within Solomon data set.

We present a summary of the results obtained in this experiment in Table 4.3. This approach overall uses 123% more robots with an overall solution quality within 1.67 of the best known ones. Our first observation is that within each category as the time windows get tighter, MCTS finds solutions using more robots resulting in large team travel distances. Secondly, we obtain worst results for the clustered test cases as multiple robots are possibly assigned to the same clusters inflicting large distance costs, and lastly we obtain best results for the R-type scenarios, the one with no clusters.

4.7.4 Analysis

When the number of robots is fixed, the random trajectories during rollouts fail to accomplish all the tasks due to the highly constrained nature of the problem. Our evaluation function punishes task allocation policies with uncompleted tasks to direct the search towards regions where the robots are likely to complete more tasks with smaller distances.

Our approach performs better for problems where n is not very small. We think that this might be due to a weakness caused by our evaluation function and our tree structure model which uses robot r_{i+1} once robot r_i returns to the depot. Although our evaluation function punishes incomplete policies with an additional distance cost, the search can be biased towards individual robot routes with smaller distances for test instances with small n. As each robot route contains many tasks for small n, MCTS can only recognize late that it cannot generate a policy which completes all the tasks through a good looking branch.

In our approach even a single sub-optimal allocation made early in the plan diminishes the quality of a fully complete policy. The main challenge is that the delayed rewards for actions resulting from earlier allocations can be understood much later. Test cases with smaller n further increase the delay of acquiring less noisy rewards due to longer routes.

4.8 Conclusions

We propose an MCTS based anytime centralized approach to solve the multi-robot task allocation problem with time windows and capacity constraints. The propose MCTS heuristic combines branch and bound pruning and a parameterized root parallelization to obtain high quality solutions while maintaining relatively low computation times. We experimentally show that our approach can generate near-optimal task allocation policies in an hour using the Solomon benchmark for vehicle routing with 100 tasks. We found solutions that are at most 1.59 away from the best-known solutions, while completing nearly all tasks. Our method maintains asymptotic completeness guarantees of the MCTS algorithm as we employ no biasing or domain-dependent heuristic during the search.

Chapter 5

Computer Narrative Generation

5.1 Introduction

Narrative is an important aspect in our lives as we can learn from narratives, convey some messages, or entertain with narratives. Other than education and training applications of narratives, as new gaming and virtual reality technologies arise, the importance of both understanding the narrative dynamics and generating narratives in a coherent and believable way increased. In this perspective, computer narrative generation has been a growing field in AI where researchers are investigating automation of narrative generation systems.

5.2 Main Results

This Chapter describes a framework for narrative generation which can learn partial narrative domain knowledge from existing books. Different from the existing work in computer narrative, our narrative planning approach provides an interactive way to create a variety of stories. Firstly, variety in obtained stories is very important due to application areas of story generators. However, to the best of our knowledge the literature has not addressed this earlier. Secondly, our narrative learning work (presented in section 5.8) address a greater challenge of understanding narrative dynamics consisting of story domain definition and action believabilities from existing story books while similar approaches constrained the training input to only very small scenarios with fixed events.

5.3 Related Work

In this section, we will review existing work in how to model and formalize narratives along with planning and learning aspects of computer narrative generation.

5.3.1 Narrative Formalization

Some efforts to automate narrative generation have focused on the important step of formalizing narratives. One example is the logic-based narrative formalization approach, Impulse, presented in [96]. Other researchers have focused on formalizing the notion of conflict for narrative generation [97], and the properties of pretend play [98]. Our work most closely follows the formalisms expressed in [99]. Like these approaches, we assume a user-defined probabilistic logic exists which defines the effects of various actions for the story domain.

5.3.2 Narrative Planning

A variety of approaches have been proposed for automating narrative planning and story creation. For example, character-centric narrative generations systems have been proposed in [100, 101] where each characters' beliefs and intentions are employed for story generation. In contrast, story-centric methods have also been proposed which reason over intentions and corresponding actions from the point of view of the audience [102]. Narrative generation has also been studied as a multi-agent planning problem [103] or by using stochastic search techniques [7]. Other authors have also looked at improving narratives by incorporating a process for influence generation [104], and by generating stories which contain morals [105].

A common application for automated story generation is that of computer games [106]. For example, the work of [107] proposes a story-making game that studies the relationship between human user behavior and narrative coherence. Likewise, a case based interactive storytelling system where a human user can contribute to the generated story is proposed in [108] which uses a knowledge base obtained from millions of stories. Another case-based system for story generation process is presented in [109]. Their framework has a sample story database and a given a new story query, they generate the overall structure of story plots with case comparisons. Computational narrative generation has also been used to create stories interactively as a game unfolds, such as in the work of [110], where a computer assisted narrative authoring system dynamically accounts for human intervention during a visual story generation.

Character-centric approaches deal with narrative generation by granting some of the narrative characters the role of story level design so that these characters ease the global planning process. For example, the work presented in [111] on Virtual Storyteller models autonomous agents and assigns them roles within the story by an external plot-agent. Storytelling has also been studied within the domain of multi-agent planning [112]. More recently, intentional planning has been combined with the multi-agent planning approaches [113].

5.3.3 Narrative Learning

Recently, many authors have looked at applying learning techniques to improve computer narrative generation. Recent examples include various approaches to the learning of drama manager strategies [114, 115, 116]. More closely related to this work, enriching story domains and learning domains have been increasingly studied recently. For example, a new technique has been proposed that enriches a given narrative domain by adding the antonymic actions of the existing actions in [117], which has also been applied to clinical narratives [118]. To decrease the authorial burden during story generation, a playable model of social interaction is proposed in [119]. The authors in [120] proposed a framework that learns a story domain from crowd-sourced stories to use in story generation. The authors in [121] proposed a method to learn character roles from unannotated folk tales. Lastly, the authors in [122] presented a data-driven learning based approach that designs different reward functions for interactive narratives.

5.4 MCTS for Story Generation

In this section, we first introduce a new story domain in which we evaluate our method. We then introduce our believability metric that guides the MCTS search, and provide a detailed explanation of our planning method.

5.4.1 Story Domain

Planning based story generation typically works over a user-specified story domain. We support a custom domain based on a simplified PDDL-type [99] of environment specification. While our approach is generic, we demonstrate it using the following crime-story inspired domain.

Our domain has three types of entities: *Actors*, *Items*, and *Places*. *Actors*, which are intended to represent people or other characters, can pick up or use various *Items*, or move to other places. Each *Item* allows different actions for an *Actor*. *Items* and *Actors* can be located at different *Places*.

Each entity has several *attributes* which allows the planner to keep track of what effect various actions have on the *Actors*, *Items*, and *Places*. For example, actors have a "health" attribute which is decreased when they are attacked. Below is an abbreviated list of the various actions allowed in our story domain followed by a brief description of its affect:

- **Move(A, P):** A moves to place P.
- **Arrest(A, B):** B's place is set to jail.
- **Steal(A, B, I):** A takes item I from B. This increase B's anger.
- **Play Basketball(A, B):** A and B play basketball. This decreases A's and B's anger.
- **Kill(A, B):** B's health to zero (dead).
- **FindClues(A):** A searches for clues at its current location
- **ShareClues(A, B):** A shares with B any clues he has found.
- **Earthquake(P):** An earthquake strikes at place P. This causes people at P to die (heath = 0), items to be stuck, and place P to collapse.

Associated with each action are methods to convert the action and corresponding *Actors*, *Items*, and *Places* to English text.

For *Actors* we have several citizens: Alice, Bob, Charlie, David, etc. There is also a detective named Sherlock, and an inspector named Inspector Lestrade. For *Places* there are several homes, recreation areas (e.g., basketball courts), and a downtown.

Items include flower vases, basketballs, baseball bats, guns and handcuffs. As discussed below, the believability of an actor taking a certain action will depend on where they are, what items they have, and their past experiences with other people.

We assume that the user specifies both an initial configuration and a goal for the story (e.g., who is in their own house, who is in downtown, where are the guns and vases). An example goal might be, "at least two people are dead and the murderer is arrested". For the purpose of running experiments, we can make the domain more complex by adding more citizens, items and places, and by changing the goal.

5.4.2 Approach Overview

Our approach uses the MCTS algorithm to find the chain of actions which accomplishes the user-defined goals with the maximum amount of believability. To apply MCTS, we must first define a function which evaluates the extent that a given story believably reaches the user's goals.

Formally, we represent a given story as a set of actions $\mathcal{A} = \{a_1 \cdots a_n\}$. We define a story evaluation function E as:

$$E(\mathcal{A}) = G(\mathcal{A})B(\mathcal{A}) \tag{5.1}$$

where $G(\mathcal{A})$ is the percentage of the user-defined goals the current story accomplishes, and $B(\mathcal{A})$ is the believability of the story as specified in Eqn 5.4.

There is a tradeoff between overall believability of a generated story and the number of goals it achieves; a story that maximizes the value of $E(\mathcal{A})$ simply finds such an optimal tradeoff which does not necessarily completes all the goals.

Importantly, this formulation allows for a series of actions that are not very believable to occur in the story if it is the only way to achieve the user's specified goals.

While $E(\mathcal{A})$ provides a natural way to evaluate a completed story, it is of limited use for partial narratives that will be encountered during a tree search. This is because until a story satisfies some of the goals, the evaluation will always be 0. We address this issue by adding a random *rollout* to the story, that is a series of random actions that is added to the partial story until all the goals are met (or until a story grows past a length threshold). We denote this randomized extension of $\mathcal{A}$ as $\mathcal{A}'$:

$$\mathcal{A}' = \{a_1, a_2, ...a_n, r_1, r_2, ...r_n\}. \tag{5.2}$$

where $r_1 \cdots r_n$ are randomly generated actions. This allows a probabilistic evaluation of $\mathcal{A}$ even when $\mathcal{A}$ does no yet reach the goal. We denote this probabilistic evaluation as E':

$$E'(\mathcal{A}) = E(\mathcal{A}'). \tag{5.3}$$

We can now formulate story generation as a Monte Carlo tree search problem. Each node in the tree will represent the complete state of the world. Each link in the tree represents one possible action from that state, and that child of the node represents the resulting world state after applying that action. The root of the tree is the initial state of the world. The MCTS algorithm proceeds by repeatedly adding one node at a time to the current tree. For each potential action, we keep track of how many times we have tried that action, and what the average evaluation was.

5.4.3 Believability

Our approach focuses on goal-oriented narrative generation. However, rather than searching to find any story which satisfies a user's goal, our approach searches for the best-possible story as evaluated by our metric. For this work, we chose a broad evaluation criteria based on how believable an action is contextually, given the current state of the story. The believability of each action is a user-defined measure on a scale from 0 to 1, which is treated as a Bayesian probability. That is, given the current state of the world, how likely it is that an event happens conditioned on the current state of the environment. For example, character A attacking character B may be more believable if A is angry. Likewise, a character arresting someone may be more believable if the character is an inspector. Some key examples from our domain are presented below.

- **Arrest(A, B)** More believable if A is an inspector. More believable if A has clues to a crime.
- **Steal(A, B, I)** More believable if item I is valuable.

- **Kill(A, B)** More believable if A is angry. More believable if A has previously killed someone.
- **FindClues(A, P)** More believable if A is an inspector or a detective.
- **ShareClues(A, B)** More believable if B is an inspector.
- **Earthquake(P)** Very low believability.

For a series of actions, we evaluate the overall believability as the product of the believability of each individual action:

$$B(a_1, a_2, ..., a_n) = \prod_{i=1}^{n} B_{a_i} \tag{5.4}$$

5.5 Enhancement to the MCTS Algorithm

In this section, we present our contributions to improve the efficiency and success of the MCTS algorithm for story generation which can be applied in other domains as well.

5.5.1 Iterative MCTS Algorithm

The MCTS algorithm keeps the entire tree in memory and can exhaust memory when exploring domains with large branching factors. This can be alleviated by pruning sections of the search tree that are unlikely to be productive. To this end, we propose an iterative approach which plans the story only one action at a time. This approach first grows the tree for a fixed number of actions. Then, only the current best action is kept, and its sibling actions' and their subtrees are pruned. This action forms the new initial condition and the tree search continues. Pseudocode for the iterative approach is presented in Algorithm 5.

As only a fixed number of nodes are added between each pruning step, the amount of memory used is bounded. We should note that this iterative approach is no longer probabilistically complete, as it is possible to prune a promising branch early on, leading to a local maxima rather than the global optimum. However, in practice we can generate high scoring narratives while using much less memory than the non-iterative approach.

Algorithm 5: Iterative MCTS for story generation and other domains.

```
Input   : Budget and max_iterations
Output: Best story
for $i \leftarrow 1$ to $max\_iterations$ do
    while $budget > 0$ do
        Node ← uctSelection(root);
        result ← rolloutStory(node);
        backpropagate(result);
        if $result > bestScoreSoFar$ then
            updateBestScore();
            saveBestStory();
        end
    end
    root ← root's most visited child;
    Prune all other subtrees;
end
return Best Story;
```

5.5.2 Biased MCTS Algorithm

Monte Carlo Tree Search can be improved by applying heuristics to help guide the search. For example, recently the authors in [123] performed selection biasing based on human-play data to generate to a player that can imitate human players. We instead heuristically learn bias from simulations. We incorporate two domain independent heuristics. For both heuristics, we keep a history table that stores average evaluation results, E', for each action (independent of it's depth in the tree). We explore two ways of using this history table: *selection biasing* and *rollout biasing.*

5.5.3 Selection Biasing

Here we modify Eqn. 2.1 to incorporate the average value for the action stored in the history table. We introduce a parameter ζ which weighs the history average value more strongly when very few (less than q) rollouts have been performed. Formally:

$$f(n) = \zeta E'(\mathcal{A}_n) + (1 - \zeta)H(n) + \sqrt{\frac{2 \ln v}{n_v}} \tag{5.5}$$

where $H(n)$ is the average value stored in history table and $\zeta = n_v/q$.

5.5.4 Rollout Biasing

In this heuristic we use the history table to bias the random rollouts. Rather than choosing pure random actions, our approach preferentially choose actions which have had a higher evaluation score as stored in the history table.

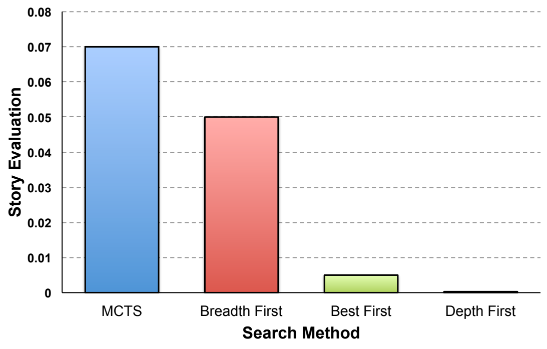
5.6 Analysis

For search method comparisons, we tested our approach on an instance of the crime story domain described above which utilized 5 actors (including 1 policeman and 1 detective), 5 places, and 5 items. The story goal is set as 2 people dead and the murderer arrested. Because each actor can use multiple items and travel to different places the resulting search space was fairly large with an average of 50 total actions available across all the actors at any given time (resulting in a search tree with an average branching factor of 50).

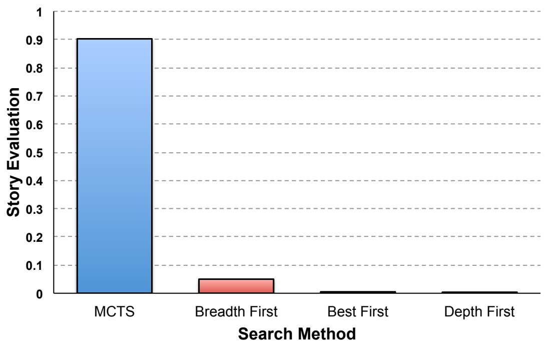
5.6.1 Search Method Comparison

We first compare our method to the traditional search algorithms of Breadth-First Search, Depth-First Search, and Best-First Search. We chose these search algorithms because, like MCTS, none of them requires a search heuristic. Furthermore, Breadth-First Search and Best-First Search algorithms are guaranteed to find an optimal solution given sufficient time and memory. Additionally, Best-First Search and Depth-First Search will explore longer paths earlier which can potentially find optimal solutions earlier in the search process. All search algorithms are implemented such that they maximize score from Eqn. 5.3. Figure 5.1 shows a comparison of the best story found by the different methods both for small and large search budgets (results averaged over 3 trials).

Depth-First search was observed to use very little memory, however, it failed to find narratives which met any goals. Best-First search suffers from delay caused by trying to accomplish the goals through a set of believable actions due to its high exploratory behavior. As a result, it tends to require higher budget to eventually find the optimal solution.



(a) Low Budget (100K Nodes)



(b) High Budget (3 Million Nodes)

Figure 5.1: **Comparison of Search Methods** Our proposed approach using Monte Carlo Tree Search (MCTS) outperforms other search techniques such as Breadth-First Search, Depth-First Search, and Best-First Search. (a) Even for a small search budget, MCTS outperforms other methods (b) The gains improve dramatically for larger budgets.

While Breadth-First search outperforms the Best-First search and Depth-First search methods, it is unable to find a believable means to achieve the goal even with a budget of several million nodes. In contrast, our MCTS approach outperforms all the other search techniques for both small and large budgets, and is able to find a high score story.

The difference in narratives generated by the various search approaches is highlighted in the illustrative sample narratives in Figures 5.3 and 5.2. These narratives are direct outputs from our code. We note that we automatically combine two consecutive related actions into a single sentence to improve readability of the narratives.

Figure 5.3 shows a sample of a high quality story, that has been generated by our MCTS algorithm. The story achieves the goals while containing several plausible actions (such as revenge killing).

Sherlock moved to Alice's House. An Earthquake occurred at Alice's House! Sherlock and Alice both died due to the earthquake.

Figure 5.2: Low Scoring Story (Score: 0.016)

Figure 5.15 shows a story found by Breadth-First search. While the story is short

Alice picked up a vase from her house. Bob picked up a rifle from his house. Bob went to Alice's house. While there, greed got the better of him and Bob stole Alice's vase! This made Alice furious. Alice pilfered Bob's vase! This made Bob furious. Bob slayed Alice with a rifle! Bob fled to downtown. Bob executed Inspector Lestrade with a rifle! Charlie took a baseball bat from Bob's house. Sherlock went to Alice's house. Sherlock searched Alice's house and found a clue about the recent crime. Bob fled to Alice's house. Sherlock wrestled the rifle from Bob! This made Bob furious. Sherlock performed a citizen's arrest of Bob with his rifle and took Bob to jail.

Figure 5.3: High Scoring Story (Score: 0.68)

and accomplishes the goal of two people being killed, it fails to achieve the more complex goal of somebody being arrested. Furthermore, the story makes use of an earthquake to reach its goals, which has a very low believability score.

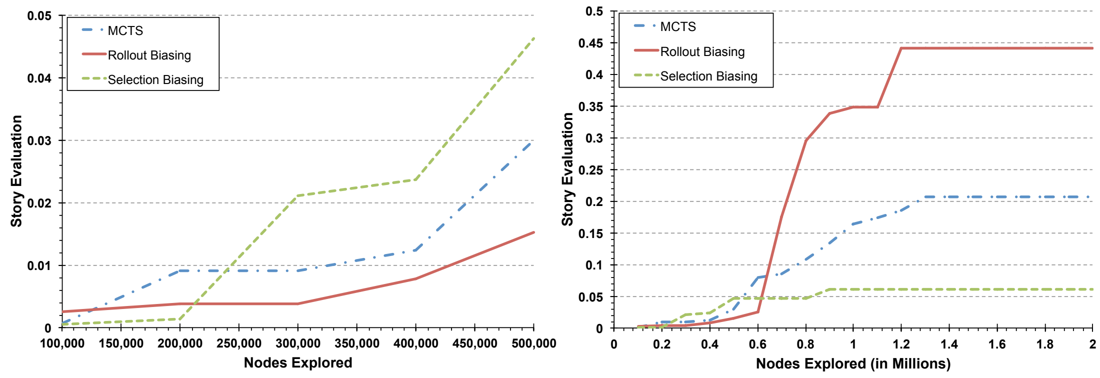
5.6.2 Heuristic Comparison

We also experiment to determine the effect of our two proposed heuristics on search performance. Figure 5.4 summarizes our results (averaged over 3 trials). For low search budgets, the selection biasing heuristic improves performance over standard MCTS (Fig 5.4a). However, this heuristic gets stuck at a local minima and fails to improve the story even with large search budgets. In contrast, the rollout biasing heuristic leads to a substantial improvement over standard MCTS for large search budgets (Fig 5.4b).

5.6.3 Large Scale Scenarios

While the vanilla MCTS approach works well, it consumes large amounts of memory. This large memory usage can restrict its applicability on very large scenes. To illustrate this limitation, we extend the crime story domain above to contain 20 actors, 7 places, and 7 items. This increases the branching factor to 150 potential actions on average.

Figure 5.5 compares standard MCTS with our iterative approach described in Algorithm 5. Importantly, the non-iterative approach fails to complete its execution when the search budget is larger than 5 million nodes. This failure happens because the non-iterative approach is using over 100GB of memory for such large search trees. In contrast, our proposed iterative approach produce better results for lower budgets, and



(a) Low Budget (b) High Budget

Figure 5.4: **Effect of Heuristics** (a) For small search budgets (<500K nodes explored) the search heuristics tested had only a moderate effect on performance. (b) For large search budgets, the advantage of the rollout biasing heuristic can be clearly seen. Additionally, while the selection bias heuristic helps with small budgets it tends to get stuck in local minima.

can run much larger budgets without failure. In fact, we were able to run with a budget over 50 million nodes on the same machine with no memory issues with iterative heuristic.

Runtime For the 5 actor story domain, our method was able to find detailed stories in under 5 seconds, and find the optimal story in less than 1 minute (using a single core on an Intel 2.2 GHz laptop processor). For the 20 actors story domain, stories took much longer to generate, though a high quality story could generally be found in under 1 hour with the iterative approach.

5.7 User-Driven Narrative Variety

We have developed an interactive framework that can generate narratives with parametric narrative goals and configurable context-sensitive believability metrics for each available action as shown in Figure 5.6. Our framework employs a graphical user interface which allows users to modify the believability of various actions by dragging several sliders. By changing believability of actions, users can influence the actions present in generated narratives via Eqn 5.4. For example, the user can generate a narrative where

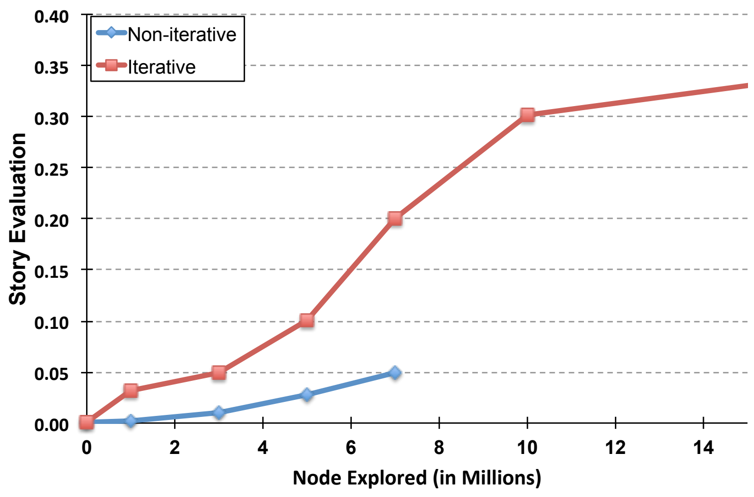


Figure 5.5: **Iterative vs Non-iterative** For very large story domains, MCTS can run out of memory trying to store the entire search tree. In the 20-person domain, the non-iterative approach could only explore trees up to 5 Million nodes before failing. Our proposed iterative approach uses tree pruning to reduce memory and can explore much large trees (producing higher value narratives).

the goal is preferentially accomplished through earthquakes by increasing the believability of the earthquake action. In this case, given the story goal of "two people dead", the resulting story is similar to the story presented in Figure 5.15.

As additional examples, we present two user-selected believability configurations in Figure 5.7. In the choice of believabilities shown in Figure 5.7(a), the user set the believability of "eartquake" and "play basketball" actions to a very low value, and the believability of "Citizen arrest" is set very high. Ideally, these choices of believabilities should generate a story where the arrest is performed by a citizen. As Figure 5.8 shows, the resulting story meets this expectation. In contrast, the second believability setup (shown in Figure 5.7(b)) has a lower belivability of citizen's arresting each other. The resulting story is shown in Figure 5.9, with the arrest ultimately made by Inspector Lestrade. Note that the two stories have similar beginnings, but they are resolved in different fashions in accordance with the user specified believabilities.

Further options are presented to the user to modify other parameters associated with our narrative generation approach. For example, selecting between iterative and non-iterative, setting the budget of maximum nodes explored, and other similar parameters.

This allows the user to control the trade-off between story quality and story generation time. The GUI along with our believability heuristic let the user be in the loop by interacting with the planner without modifying the MCTS side of the framework.

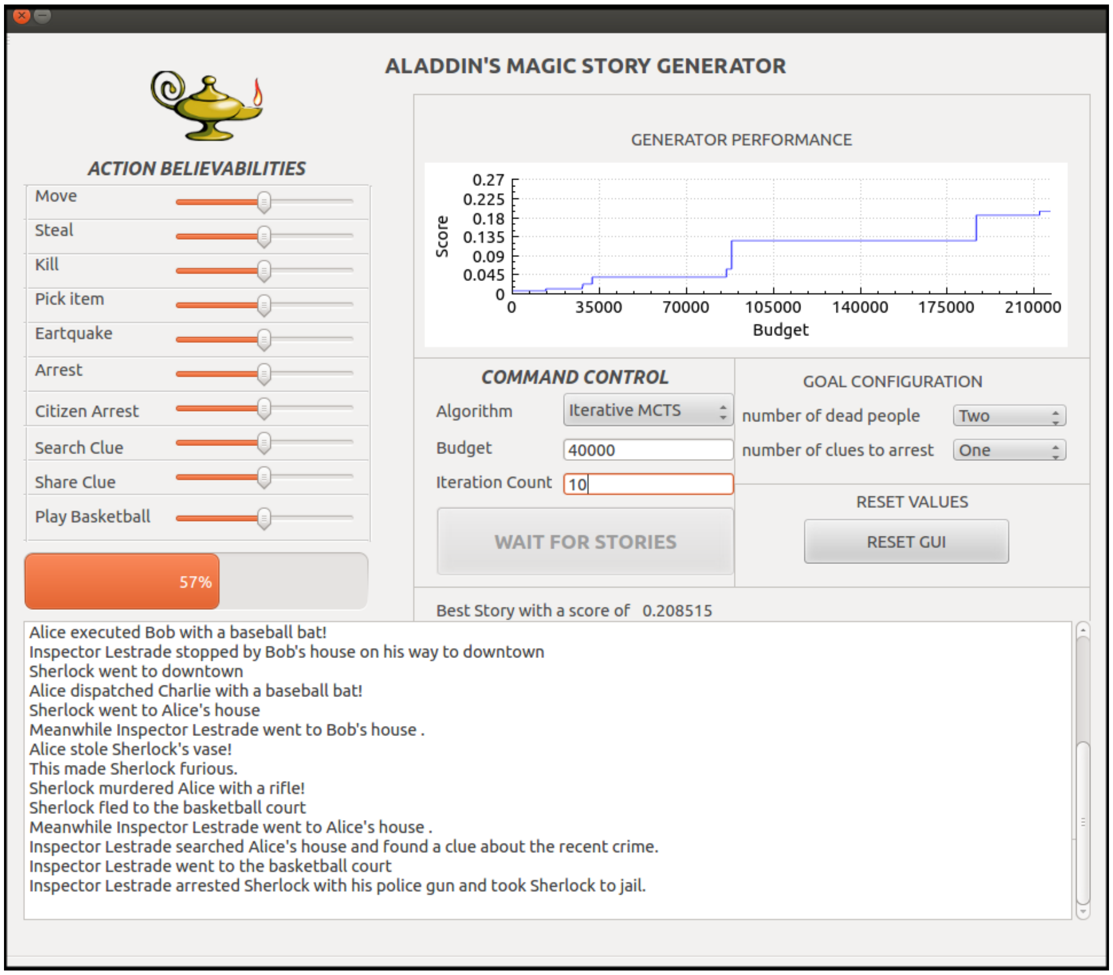
5.8 Learning Narrative Planning Domains

Our initial work [7] on story generation assumed that a high level domain is manually authored and input to the MCTS planner. However, this step is time-consuming and it must be repeated when the narrative domain is changed. In this part of this thesis, we will present our work on learning narrative domains from existing story books that can be obtained from project Gutenberg.

The need for a user to manually author high-level story domain creates a bottleneck for the automation of narrative generation. We propose a data-driven narrative learning method that employs a Bayesian inference approach to learn high-level story domains from collections of existing stories. Our method generates both story domain parameters and the believability of various actions that may be taken on these learned story parameters. Our method is semi-supervised, largely automating the process of domain generation and story creation, while still allowing for easy feedback from users to quickly edit, augment, or delete any unsatisfying aspects of the learned domains. The resulting story domains are unique, reflecting key aspects of the source text used in training. As a proof of concept, we employ our learned story domain knowledge to automatically generate narratives.

5.8.1 Approach Overview

We formulate narrative domain inference problem as a learning problem. Besides the set of actions (which are assumed to be given), we define a story domain as a set of actors who perform these actions, a set of items which the actors can use, and a set of places over which the story unfolds. Additionally, I require domains to incorporate some notion of believability which captures how likely a given person is to perform a given action with a given item (e.g., killing with a rifle should be much more likely than killing with a handkerchief). Our goal then, is to learn all of these elements from source stories.

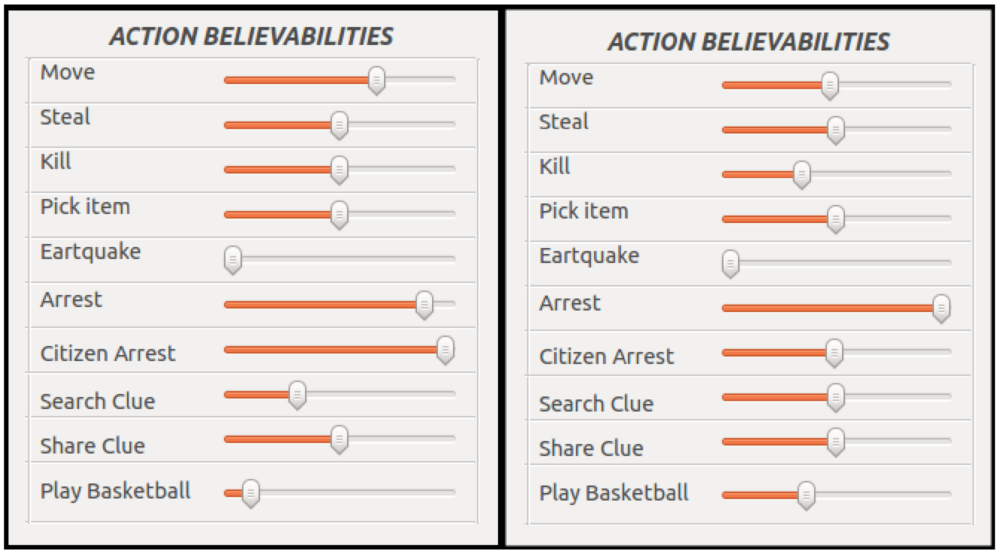


Algorithm: Iterative MCTS
Budget: 40000
Iteration Count: 10
number of dead people: Two
number of clues to arrest: One

Best Story with a score of 0.208515

Alice executed Bob with a baseball bat!
Inspector Lestrade stopped by Bob's house on his way to downtown
Sherlock went to downtown
Alice dispatched Charlie with a baseball bat!
Sherlock went to Alice's house
Meanwhile Inspector Lestrade went to Bob's house .
Alice stole Sherlock's vase!
This made Sherlock furious.
Sherlock murdered Alice with a rifle!
Sherlock fled to the basketball court
Meanwhile Inspector Lestrade went to Alice's house .
Inspector Lestrade searched Alice's house and found a clue about the recent crime.
Inspector Lestrade went to the basketball court
Inspector Lestrade arrested Sherlock with his police gun and took Sherlock to jail.

Figure 5.6: Aladdin's Magic Story Generator for Interactive Narrative Generation. Users can modify believabilities of various actions, set story goals, select a planning strategy, and choose a planning budget. As the generation process unfolds, the best story found so far is displayed along with a graph of the story evaluation score progress.



(a) Citizen arrest configuration (b) Officer arrest configuration

Figure 5.7: Users can set believability of actions to generate diversity in generated stories.

Alice got a vase from her house. Bob picked up a rifle and a baseball bat from his house. Sherlock stopped by the basketball court on his way to Alice's house. Charlie went to Alice's house. Charlie took Alice's vase! This made Alice furious. Alice stole Charlie's vase! This made Charlie furious. Alice killed Charlie with a flower vase, very interesting! Bob went to Alice's house. Meanwhile Sherlock went to the basketball court. Inspector Lestrade stopped by Bob's house on his way to Alice's house. Alice fled to the basketball court. Bob searched Alice's house and found a clue about the recent crime. Sherlock went to Bob's house. Meanwhile Bob went to downtown. Alice stopped by Alice's house on her way to Bob's house. Inspector Lestrade went to downtown. Meanwhile Bob went to Alice's house. Alice executed Sherlock with a flower vase, very interesting! Bob went to Bob's house. Bob arrested Alice with his baseball bat and took Alice to jail.

Figure 5.8: Story generated from believability setup shown in Figure 5.7(a). In this configuration, the believability of "citizen arrest" action is set high, resulting Bob arresting the murderer.

Alice secured a vase from her house. Bob secured a baseball bat and a rifle from his house. Bob went to Alice's house. Alice pilfered Bob's rifle! This made Bob furious. Inspector Lestrade went to the basketball court. Bob slayed Alice with a baseball bat! Bob slayed Charlie with a baseball bat! Bob fled to the basketball court. Meanwhile Sherlock went to Bob's house. Sherlock went to Alice's house. Meanwhile Inspector Lestrade went to Bob's house. Inspector Lestrade went to Alice's house. Inspector Lestrade searched Alice's house and found a clue about the recent crime. Inspector Lestrade went to the basketball court. Meanwhile Bob went to Alice's house. Bob stopped by the basketball court on his way to downtown. Bob and Sherlock both went to the basketball court. Sherlock picked up a basketball from the basketball court. Inspector Lestrade arrested Bob with his police gun and took Bob to jail.

Figure 5.9: Story generated from believability setup shown in Figure 5.7(b). In this configuration, the Inspector Lestrade arrests the murder due to low believability of "citizen arrest" action.

We group together several stories which share common source materials in order to increase the amount of training data for each domain. Specifically, I work with three classes of stories, i.e. Children Fairy Tale, Detective, and Shakespeare (all obtained from Project Gutenberg, a repository of public domain books (www.gutenberg.org)).

For each story dataset, the corpus vocabulary is denoted by V, and the set of all sentences as $\hat{S}$. We define S_i^j as the jth word of the ith sentence, and $|S_i|$ denotes the number of words in ith sentence. We select a subset of sentences denoted by S that contains all of the sentences including our action words:

$$S = \{S_i \in \hat{S} \mid \exists j : S_i^j \in A\}. \tag{5.6}$$

We treat the word to category assignment task as a probabilistic categorization problem. Specifically, our goal is to estimate the probability that each word in the vocabulary belongs to one of categories of (person, place, item, or no category). We do this by estimating a probability distribution, P, for each word in V over all categories:

$$P_i = \begin{bmatrix} p(V_i = Person) \\ p(V_i = Item) \\ p(V_i = Place) \\ p(V_i = Other) \end{bmatrix}. \tag{5.7}$$

Given a prior for each term in our vocabulary, the role of the data-driven learning process is to refine these priors based on the input corpus stories. Our approach centers around the use of potential sentence templates that match each action. Broadly speaking, we develop an iterative approach with two alternating update steps. First, for each sentence, the currently estimated word category probabilities are used to compute the most likely action template that sentence belongs to. Second, we update the word category probabilities based on these inferred templates for each sentence. These category probabilities will, in turn, affect the template matching process on subsequent iterations, and so on. The process is run iteratively until convergence, or some other stopping criteria is met (such as a maximum number of iterations).

Once our approach has categorized each term, we then seek to quantify the believability of the various terms in the context of various actions. we employ a two step process to learn the distribution of categories for each term. First, we utilize a pre-existing semantic knowledge base to obtain domain independent initial probabilities of possible categorizations of each term. Second, we use our story corpus to refine these initial probability estimates following an EM-style approach of iterative refinement.

We preprocess story corpuses by splitting into sentences, removing common words, and lemmatizing words using Natural Language Toolkit (NLTK) [124]. After the preprocessing, our algorithm proceeds with the following five steps:

- Split corpus into sentences using NLTK sentence splitter.
- Remove common words (e.g. the, of, and in).
- Filter out sentences not containing any of our action words.
- Lemmatize words using NLKT lemmatizer which reduce words to a more general form (e.g. dogs become dog).

- Generate a local dictionary from all words in our filtered and lemmatized sentences.

5.8.2 Prior Knowledge Inference

We primarily employ ConceptNet [125], an automatically generated semantic knowledge base, that maps words to each other with a set of predefined relationships. We present an extracted subset of the ConceptNet graph in Figure 5.10. We selected a salient subset of these relationships to infer prior probabilities of words. We employed relative frequencies of these salient relationships to initialize prior probabilities. These frequencies are regularized and then normalized to produce an estimated initial distribution of categories for each word. Regularization reduces the risk of becoming over confident in the prior values.

Algorithm 6: Prior Generation from ConceptNet

```
for $i=1:|V|$ do
    $P_i = [0 ; 0 ; 0 ; 0 ]$
    Rel = Query $V_i$ from ConceptNet
    if $Rel[begin]$ is $V_i$ then
        if $Rel[end] \in PersonFlag$ then
            $P_i[1]++$
        end
        if $Rel \in ItemFlag$ then
            $P_i[2]++$
        end
    end
    else
        if $Rel \in PlaceFlag$ then
            $P_i[3]++$
        end
    end
    if $Rel \in OtherFlag$ then
        $P_i[4]++$
    end
    $P_i = P_i + c$          // Regularization
    $P_i = \frac{P_i}{\|P_i\|_1}$          // Normalization
end
```

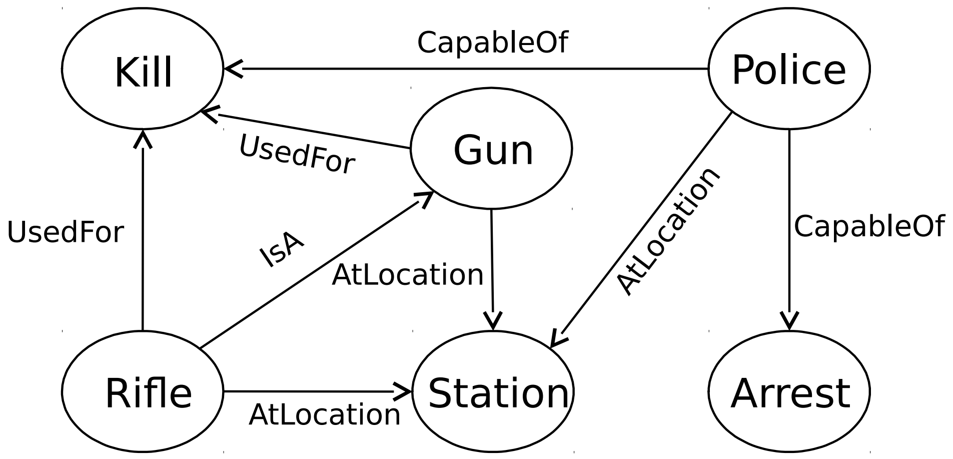


Figure 5.10: An example subgraph of ConceptNet is presented. Words are related to each other by a set of predefined relations which provides a rich information setting.

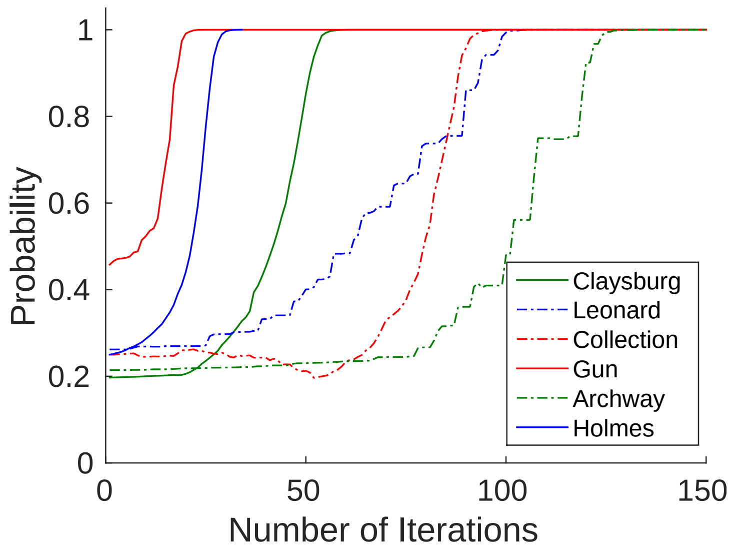


Figure 5.11: The evolution of category probability distribution for a selected set of words is presented. Blue lines refer to *Actors*, red to *Items*, and green to *Places*.

5.8.3 Action Template Matching

For each action, we derive a corresponding set of action templates that capture the typical usage of the action. For example, one of the templates for the *Move* action is {Person, Place}, which would match the sentence “Bob went to the school”, whereas the *Arrest* action has the template of {Person, Person, Item} and would match the sentence “Alice used her gun to place Bob under arrest”.If a sentence has multiple actions, it is arbitrarily assigned to one action category.

To improve performance, we restrict template length to four. In addition to restricting template length, we pad short templates with an “other” category, such that all templates are of length four. When sentences are longer than templates, we select a subset of words to match to the template length. Through repeated iterations of our algorithm, many different word combinations will be tried for each template.

5.8.4 Template Weight Computation

We use a bag-of-words model to calculate the degree to which the template fits the words. That is, we find the order-independent permutation of words for which the categories best match the templates. The template score is computed by the product of percentage of each word’s distribution in the appropriate category for the template.

5.8.5 Word Probability Update

To update the probability a word is a particular type, we sum the probability of each template it appears in as that word type. we then normalize these such that the sum over all category types for a particular term is 1. As our data is inherently noisy, we don’t replace our prior with this new value, we instead firstly incorporate a dynamic regularization term to smooth out the computed posterior category probabilities. This smoothing terms depends on two factors that determine the observation confidence. First, the larger the template score, the more confident our approach is that random words chosen are the best ones fitting to the sentence’s action template. Secondly, we look at ratio between the number of times a word appears associated with one of our known actions, versus its occurrence in the entire corpus. We denote this ratio as F. Words, which typically are associated with our actions will therefore converge more

Algorithm 7: Domain Learning Algorithm

```
while Not converged do
    for i=1:|S| do
        for template ∈ Templates($S_i$) do
            if $|S_i| > |template|$ then
                words = Randomly Pick from $S_i$
            end
            else
                words = $S_i$
            end
            permutations = permute(template)
            for permutation ∈ permutations do
                $f_i = \prod_j P(S_i^j = \text{permutation}_j)$
                Add $f_i$ to posterior matrix
            end
        end
        Smoothly Normalize Posterior (Eqn. 5.8)
        Update Category Estimates (Eqn. 5.9)
    end
end
```

quickly than those which are not. This process is captured by the following equation, which also includes a user-tunable parameter k:

$$\hat{P} = \hat{P} + k\frac{F}{max(f_i)}. \tag{5.8}$$

After applying this equation, the probabilities are normalized across the categories so as to sum to one. We now use this newly inferred category distribution to update the prior distribution via a multiplicative update as follows:

$$P = P * \hat{P}. \tag{5.9}$$

Algorithm 7 outlines the entire category learning processes.

	Steal	Kill	Arrest	Grab	Search	Marry
Holmes	0.86	0.11	0.37	1.00	1.00	0.21
John	1.00	1.00	0.39	0.16	0.33	1.00
Police	0.02	0.16	1.00	0.20	1.00	0.00

Table 5.1: Inferred believabilities for actor-action pairs for a subset of actions for the detective story domain.

5.8.6 Believability Inference

Once the story domains are created, our approach infers approximate believability parameters by using a simple context information. To generate believabilities, the pairwise co-occurrence of words is considered. The more frequently words occur in the same sentence, the more believable that pair is. We compute the item-action believability by dividing each item-action co-occurrence by the maximum item-action co-occurrence.

Formally, let I keep the co-occurrence count of items for all actions in A, and let B be the item-action believability matrix of size the same size I where $B(i, j)$ denotes the believability of using item i with action j.

$$B(i,j) = \frac{I(i,j)}{max(I(:,j))}. \tag{5.10}$$

where the item-action co-occurrences are divided by the maximum item-action co-occurrence. We can compute the actor-action believabilities similarly using the actor-action co-occurrences. To approximately compute the believability of an actor i performing action j with item k, we simply multiply the actor-action believability with the item-action believability.

Additionally, we employ the same metric as Eqn. 5.10 to find initial locations for actors and items in the story environment. We normalize the co-occurrence rates of the actors and items with places to calculate the probability of an actor or item appearing in a particular place.

A sample of the actor-action believability matrix is shown in Table 5.1. Once again, our approach creates some intuitive results, such as *Holmes* and *Police* searching frequently, but it also generates some incorrect results, such as *Holmes* stealing frequently.

5.9 Domain Learning Results

We present the inferred believability matrix for item-action relationship in Table 5.3. Our results are mostly intuitive, with revolvers being most likely used to kill or to arrest, while games are very likely used for play. However, there are some small issues, such as money being used to arrest being relatively believable. We experimented our approach on the following three domains collected from the Project Gutenberg. We report both learned domain knowledge and results on classification errors for each story corpus.

- **Fairy Tale** A collection of children's fairy tales
- **Detective** A collection of Sherlock Holmes novels
- **Shakespeare** A collection of Shakespeare plays

Figure 5.12c shows the evolution of the classification and misclassification rates over time for the Shakespeare domain. A word is only considered classified if it's probability for the correct category is greater than 50%. It is considered misclassified if the probability is greater than 50% in an incorrect category. As the figure shows, the classification rate is much higher than the misclassification rate. In addition, the classification rate converges much quicker than the misclassification rate. This allows us to limit the number of iterations to reduce the misclassification rate, with little effect on the classification rate.

For runtime, a single iteration with approximately 10,000 sentences takes about 15 seconds (using a single core on an Intel 2.2 GHz laptop processor).

We present the learned children story domain in Table 5.2, detective domain in Table 5.4, and Shakespeare domain in Table 5.5 for the corpus we studied. Words are listed in the order at which they converge to a single category. At this point, we allow human intervention, where the user can select and-or remove some of the words from the converged words lists. Finally selected words will be fed into the story planning algorithm. We should note that pronouns are removed from the actor words as they are the fastest converging words across all domains.

Figure 5.11 shows the results of the category learning process in action, presenting the convergence of different words across multiple iterations over the *Detective* training

	Learned Children Story Domain
Actors	King - Prince - Dog - Jack - Princess Bride - Lass - Queen - Calf - Hero
Items	Game - Heart - Sword - Wood - Ball Harp - World - Bone - Money - Arrow
Places	House - Country - Forest - Room - Tree Road - Palace - Path - Street - Sea

Table 5.2: Domain knowledge learned for the Children story domain. The words are listed by the order they converged.

	Steal	Kill with	Arrest with	Play with	Grab
Revolver	0.08	0.72	1.00	0.02	1.00
Knife	0.03	1.00	0.04	0.07	0.95
Necklace	0.22	0.12	0.11	0.04	0.59
Money	1.00	0.04	0.31	0.23	0.09
Game	0.04	0.05	0.00	1.00	0.05

Table 5.3: Inferred believabilities for item-action pairs for a subset of actions for the detective story domain.

story corpus. Some terms, such as *Gun*, have very good values coming from our prior initialization step (Alg. 6) . These terms very quickly converge to the correct value (100% item categorization). In contrast, some words such as *Holmes* have little or no relevant relationship in ConceptNet, so start with a nearly uniform prior (25% item, 25% actor, 25% place, 25% other). However, these terms can still be seen to converge to the correct categories. Words like *Holmes* that appear frequently in our corpus, can often converge faster than those that appear infrequently.

	Learned Detective Story Domain
Actors	Aston - Girl - Police - Murderer - Heredith Shepley - Glenthorpe - Holmes - John - Prisoner
Items	Card - Diamond - Money - Heart - Evidence Knife - Revolver - Fortune - Necklace - Handkerchief
Places	Room - Street - Door - Marbury - Office Gate - Hotel - Village - Air - London

Table 5.4: Domain knowledge learned for the Detective story domain. The words are listed by the order they converged.

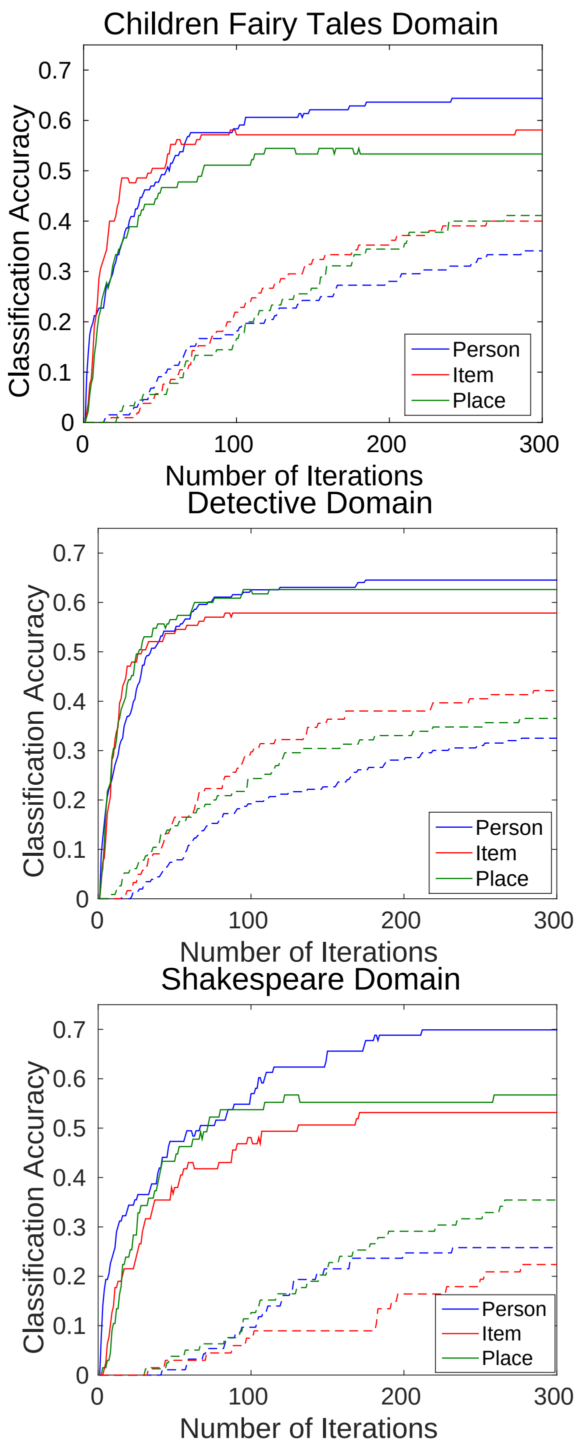


Figure 5.12: Classification and misclassification(dashed lines) accuracies are presented for categories of actors, items, and places.

	Learned Shakespeare Story Domain
Actors	Thee - Lord - Sir - Ceasar - King Faith - Maid - Mistress - Villain - Troyans
Items	Hearth - Fortune - Soul - Music - Sword Ducat - Letter - Instrument - Spirit - Wager
Places	Heaven - Paris - House - Country - Lobby University - Street - Ere - Gloucestershire - Nature

Table 5.5: Domain knowledge learned for the Shakespeare story domain. The words are listed by the order they converged.

5.10 Story Generation from the Learned Domains

We employ Monte Carlo Tree Search for generating story plans as proposed in [7]. However, we use the learned story domain parameters for the sets of actors, items, and places. Furthermore, we infer two believability matrices, item-action and actor-action, to employ during planning for more believable stories. Also, in this work, we relax the action-item matching constraint proposed in the aforementioned study and instead let the planning phase resolve it. Since we don't predefine which items can be used for which actions, we have a much bigger search space for planning phase. However, MCTS is shown to perform well for large search spaces and we present some examples of generated stories in Figures 5.16 and 5.14.

We present a story generated with very low search time in Figure 5.15 where the *kill* and *arrest* actions are performed with items whose action-item matrix scores are low, therefore the resulting story is less believable.

Princess went to the forest. Prince and Princess fell in love with each other. [...] Princess killed King with the treasure at the castle. [...] Prince and Princess got married!

Figure 5.13: A low quality story generated in a second from the Fairytales story domain with inferred domain knowledge.

We employed detective and children fairy tale stories as corpus to generate different stories. For detective story domain, we defined the goal as follows: one actor was killed, and murder was arrested. However, over time planning yields a converged story as shown in Figure 5.16 where actions are performed with items such that action-item

King picked up treasure and sword at the castle. Princess went to the forest. King went to the forest. King and Princess fell in love with each other. Prince stole King's treasure. King killed Prince with the sword at the forest. Princess witnessed the crime. King and Princess got married!

Figure 5.14: A converged high quality story generated in less than a minute from the Fairytales story domain with inferred domain knowledge.

values are high, resulting a more believable story.

John picked up the game at the house. John killed Holmes with the game at the house. Policeman picked up the necklace at the station. Policeman went to the house. Policeman arrested John with the necklace at the house.

Figure 5.15: A low quality story generated in a second from the Detective story domain with inferred domain knowledge.

John picked up the money and the knife at the office. Police went to the street. Police picked up the revolver at the street. Police went to the office. Holmes went to the office. Holmes stole John's money. John killed Holmes with the knife at the office. Police witnessed the crime. Police arrested John with the revolver at the office.

Figure 5.16: A converged high quality story generated in less than a minute from the Detective story domain with inferred domain knowledge.

For this goal, our search approach finds the low quality story excerpted in Figure 5.13 where the story is less believable because *Princess* is employing the *treasure* to kill *King*, where both the believability of killing with treasure and Princess' being the murderer have low believability scores.

For the children story domain, we defined the story goal as follows: one actor died, one item was stolen, and a marriage took place. With several more seconds of search, the story in Figure 5.14 is generated. Here, the story is of a much higher quality accomplishing all the story goals and supports our inferred believabilities through action-item and action-actor matrices.

We presented a data-driven approach to extract salient narrative domain knowledge from story books. We also presented a method for employing this knowledge coherently with different actions. Our approach is able to learn different domain parameters from different story corpuses. We also used the learned word probabilities in our story planning system to infer item-action and actor-action believabilities. We let a human user intervene and edit story domain parameters to choose what will be used in the planning phase. Finally, we presented rendered stories showing that converged stories get more and more believable.

5.11 Conclusions

We have presented a framework capable of generating believable narratives which satisfy user-defined goals from large story domains. By using Monte Carlo Tree Search, our method is able to balance exploiting the most promising branches along with exploring other potentially good choices at each level of the tree. The resulting framework generates complex, believable narratives with only a few seconds of computation time for small domains, and a few minutes for larger ones. We also introduced a user-friendly tool that can be used by authors and teachers to generate full or partial narratives for specific scenes. We further presented an inference technique to better automate the computer narrative generation. Our approach is able to learn different domain parameters from different existing story corpuses obtained from project Gutenberg. We also used the learned word probabilities in our story planning system to infer item-action and actor-action believabilities. While we are able to generate believable stories from our corpuses, we still require some manual authoring of domain knowledge.

In Chapter 6, we study another closely related PCG problem, i.e. Sokoban puzzle generation which has application areas such as games and education similar to computer generated stories.

Chapter 6

Sokoban Puzzle Generation

6.1 Introduction

Puzzle games play an integral role in entertainment, intellectual exercise, and our understanding of complex systems. Generating these puzzles automatically can reduce bottlenecks in design, and help keep games new, varied, and exciting. Furthermore, generating a variety of puzzles with controlled difficulty allows us to custom tailor game experiences to serve a much wider population, including those with little previous video game or puzzle solving experience.

We study the above challenges within the context of the puzzle game of Sokoban. Developed for the Japanese game company *Thinking Rabbit* in 1982, Sokoban involves organizing boxes by pushing them with a player controlled agent on a discrete grid board. We propose a method that automatically generates Sokoban puzzles. In order to support the dynamic needs of a large variety of users, our system needs to address several challenges inherent in the field of puzzle generation. These include the speed of the system, supporting on-demand puzzle generation, and producing a variety of puzzles. These properties support a range of player skills, and are key factors in keeping player experiences engaging.

6.2 Main Results

This chapter describes our method for Sokoban puzzle generation. Current methods for procedural Sokoban puzzle generation tend to use exponential time algorithms that require templates or other human input. Achieving the goal of a fast, varied, and predictive system requires overcoming several challenges in automating the understanding of puzzle difficulty and generating puzzles of desired difficulty levels. Our work moves towards addressing these challenges via the following contributions:

- *Assessing level difficulty.* We utilize a user study to annotate the perceived difficulty of an initial set of Sokoban puzzles.
- *Learning features predictive of difficulty.* We use statistical analysis to infer features that are predictive of puzzle difficulty and are efficient to compute.
- *Generating varied, solvable puzzles that optimize learned features.* We formulate puzzle generation as an MCTS optimization problem, modeling the search tree structure such that puzzles are generated through simulated game play.

The result is an anytime algorithm that produces levels of varying difficulty that are guaranteed to be solvable.

6.3 Background

There have been many applications of Procedural Content Generation (PCG) methods to puzzle games, such as genetic algorithms for *Spelunky* [126], MCTS based *Super Mario Bros* [33], map generation with Markov chains [127, 128, 129], and regular expression based level generation [130]. Other approaches propose search as a general tool for puzzle generation [131], and generation of different start configurations for board games to tune difficulty [132]. Some even dynamically adapt to player actions [133]. The authors in [134] propose an answer set programming based paradigm for PCGs for games and beyond. A recent approach parses game play videos to generate game levels [135]. The authors in [136] proposed a probabilistic approach to better understand the game-space that can be used during design process. We refer the reader to the survey [137] for a more detailed overview.

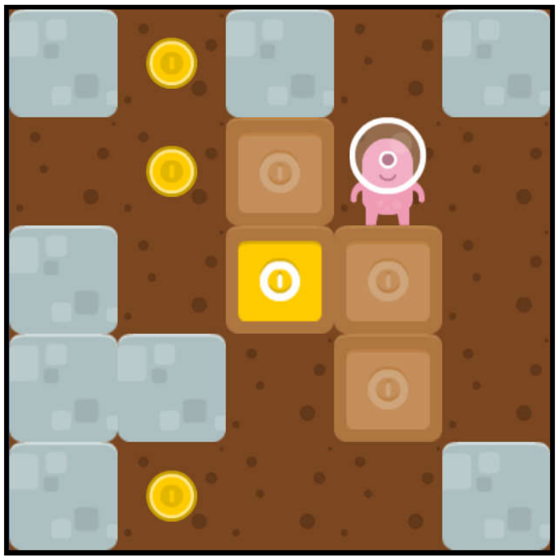


Figure 6.1: A high scoring 5x5 Sokoban puzzle generated by our method. The goal is to move the agent to push boxes (brown squares) so that all goals (yellow discs) are covered by the boxes. Yellow filled boxes represent covered goals. Obstacles (gray squares) block both agent and box movement

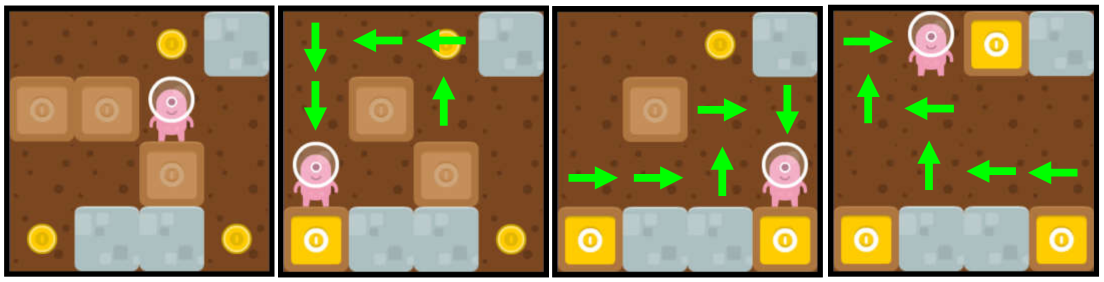


Figure 6.2: A generated Sokoban puzzle with solution (score = 0.31).

6.3.1 Sokoban Puzzle

The Sokoban game board is composed of a two-dimensional array of contiguous tiles, each of which can be an obstacle, an empty space, or a goal. Each goal or space tile may contain at most one box or the agent. The agent may move horizontally or vertically, one space at a time. Boxes may be pushed by the agent, at most one at a time, and neither boxes nor the agent may enter any obstacle tile. The puzzle is solved once the agent has arranged the board such that every goal tile also contains a box. We present an example solution to a Sokoban puzzle level in Figure 6.2.

Previous work has investigated various aspects of computational Sokoban including automated level solving, level generation, and assessment of level quality.

Previously proposed frameworks for Sokoban PCG involve creating many random levels and analyzing the characteristics of feasible solutions. However, solving Sokoban puzzles has been shown to be PSPACE-complete [138]. Some approaches have focused on reducing the effective search domain [139]. Recently, Pereira et al. [140] have proposed an approach for solving Sokoban levels optimally, finding the minimum necessary number of box pushes. Pure MCTS has been shown to perform poorly for solving Sokoban puzzles [141].

While there have been many attempts for solving Sokoban puzzles, the methods for their procedural generation are less explored. To the best of our knowledge, Murase et al. [142] proposed the first Sokoban puzzle generation method which firstly creates a level by using templates, and proceeds with an exponential time solvability check. More recently, the authors [143] proposed a similar approach, using templates for empty rooms and enumerating box locations in a brute-force manner. Their method can generate compelling levels that are guaranteed to be solvable. However, the run-time is exponential, and the method does not scale to puzzles with more than a few boxes.

There have been several efforts to assess the difficulty of puzzle games. One example is the very recent work by [144], which combines features common to puzzle games into a difficulty function, which is then tuned using user study data. Others consider Sokoban levels specifically, comparing heuristic based problem decomposition metrics with user study data [145], and using genetic algorithm solvers to estimate difficulty [146]. More qualitatively, Taylor et al. [147] have conducted a user-study and concluded that computer generated Sokoban levels can be as engaging as those designed by human experts.

6.4 Anytime Formulation with MCTS

One of the challenges for generating Sokoban puzzles is ensuring solvability of the generated levels. Since solving Sokoban has been shown to be PSPACE-complete, directly checking whether a solution exists for a candidate puzzle becomes intractable with increasing puzzle size. To overcome this challenge, we exploit the fact that a puzzle can be generated through simulated gameplay. To do so, we decompose the puzzle generation problem into two phases: puzzle initialization and simulated gameplay. Puzzle initialization refers to assigning the box start locations, empty tiles, and obstacle tiles. Simulated gameplay consists of a simulated player performing sequences of box pushes to determine goal locations. As the agent moves around during the simulation, it pushes boxes to different locations. A final snapshot of the resulting board configuration defines goal locations for boxes.

We apply MCTS by formulating the puzzle creation problem as an optimization problem. The main reasons for using MCTS to generate Sokoban puzzles include its success in problems with large branching factors, the anytime property, and the search structure that guarantees solvability. As discussed above, the search tree is structured such that the game can be generated by simulated gameplay. The search is conducted over both puzzle initializations and gameplay actions. Because the simulated gameplay is conducted using Sokoban game rules, invalid paths are never generated. In this way, our method is guaranteed to generate only solvable levels.

Anytime algorithms return a valid solution (if a solution exists) even if it is interrupted at any time. Given that our problem formulation is completely deterministic, MCTS can store the best found puzzle after rollouts during the search and optionally halt the search at some quality threshold. This behavior also enables us to create many puzzle levels from a single MCTS run with monotonically increasing scores.

6.4.1 Action set

Our search tree starts with a board fully tiled with obstacles, except for the agent start position. Initially, the following actions are possible at any node in the search tree:

1. *Delete obstacle*: An obstacle that is adjacent to an empty space is replaced with an empty space. This progressive obstacle deletion prevents boards from containing

unreachable regions.

2. *Place box*: A box may be placed in any empty tile.

3. *Freeze level*: This action takes a snapshot of the board and saves it as the start configuration of the board.

After the *Freeze level* action is chosen, the action set for descendant nodes of the frozen puzzle node is replaced by two new actions:

1. *Move agent*: This action moves the agent on the game board. The agent cannot move diagonally. This action provides the *simulated gameplay* mechanism, where the boxes are pushed around to determine goal positions.

2. *Evaluate level*: This action is the terminal action for any action chain; it saves the rearranged board as the solved configuration of the puzzle (i.e. current box locations are saved as goal locations).

These two action sets separate the creation of initial puzzle configurations (actions taken until the level is frozen) from simulated gameplay (agent movements to create goal positions). A key property of this two-phase approach is that it maintains the uniqueness of states throughout the tree; no two nodes represent the same board layout and agent path. This helps improve efficiency by reducing redundant search paths. Since the agent can move indefinitely back and forth, we enable the *Evaluate Level* action at all the branches where the agent can move.

Once the *Evaluate level* action is chosen, we apply a simple post-processing to the board in order to remove elements that are known to be uninteresting. In particular, we turn all boxes that are never pushed by the agent into obstacles as this does not violate any agent movement actions. We also replace boxes that are pushed only once with an empty space (and delete the associated goal). This post-processing is performed before evaluating the level.

A critical component of our MCTS formulation that has yet to be addressed is the evaluation function. As MCTS is an optimization algorithm, we must provide it with an objective function that describes the desired properties of candidate puzzles. To accomplish this, the function maps from candidate puzzles to a score dependent

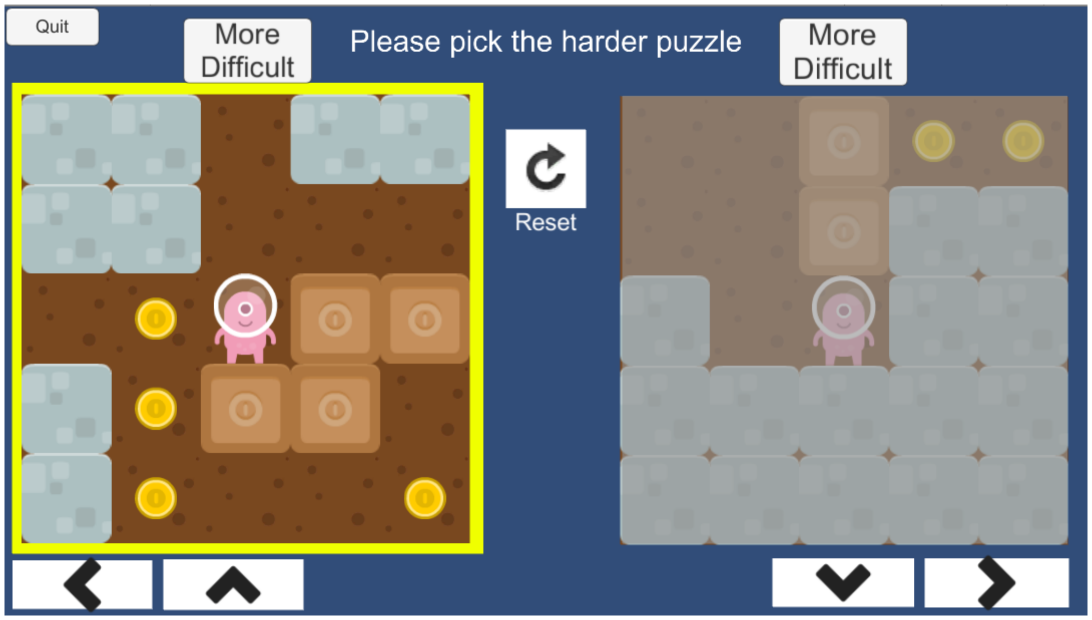


Figure 6.3: Our user study application, which presents pairs of puzzles to subjects and asks them to identify the one that is more challenging. Subjects were able to play each level presented as much or as little as desired before making a decision.

upon how difficult or interesting the puzzle is. This involves finding features of Sokoban puzzles that can be computed quickly and are predictive of puzzle difficulty. We propose a data driven way to produce such a function in the following section.

6.5 Data-Driven Evaluation Function

Our goal is to generate levels which are not only solvable, but also engaging or difficult. We address this with a data-driven approach. First, we perform a user study analyzing the perceived difficulty of Sokoban puzzles. We then use this analysis to propose and validate new features estimating the level difficulty. Finally, we utilize these inferred features in our MCTS framework to efficiently generate Sokoban puzzles.

6.5.1 Estimating Perceived Difficulty

One challenge in taking a data-driven approach for difficulty estimation is the lack of large datasets of Sokoban puzzles that have known difficulty. The purpose of our user study was to create such a dataset. To facilitate this, we developed a custom Sokoban player application where users are shown two levels and asked to select the one that is more difficult (Figure 6.3). The users can switch between shown levels anytime and decide on the harder level without needing to complete the games. This application was placed on an Android tablet and users were allowed to rate as many puzzle pairs as they liked.

We collected user ratings for 120 preexisting puzzles including both human-designed puzzles obtained from [148] and computer generated ones obtained from [8]. Over the course of two weeks, we had approximately 30 participants provide 945 pairwise comparisons.

In order to estimate the perceived difficulty of each puzzle, we employed the TrueSkill Bayesian skill estimation system [149]. Briefly, each puzzle's estimated difficulty is represented by a Gaussian, with a mean at the estimated difficulty score, and a standard deviation representing the uncertainty in the estimation. Each time a puzzle is decided to be more difficult than another, its mean (estimated difficulty) increases and the other puzzle's mean decreases. The estimated uncertainty decreases as more ratings are gathered for each puzzle. These TrueSkill means typically range from 0 (least difficult) to 50 (most difficult), and are referred to in this paper as *Perceived Difficulty.*

6.5.2 Feature Analysis

The comparison results from the user study were compiled and used to annotate the puzzles with their perceived difficulty. Because the evaluation function is invoked for every *Evaluate level* action of MCTS, we restricted our search for features to only those that were efficient to compute. In particular, we do not include features based on an optimal solution, as finding an optimal solution is a PSPACE-complete task.

We tested several features for correlation with perceived difficulty, including:
Metrics analyzing the layout of obstacles and free space

- *Tile Mixing.* The number of free space tiles next to obstacle tiles, and obstacle tiles next to free space.
- *3x3 Block Count.* The number of tiles not in a 3x3 block of solid obstacles or open space.

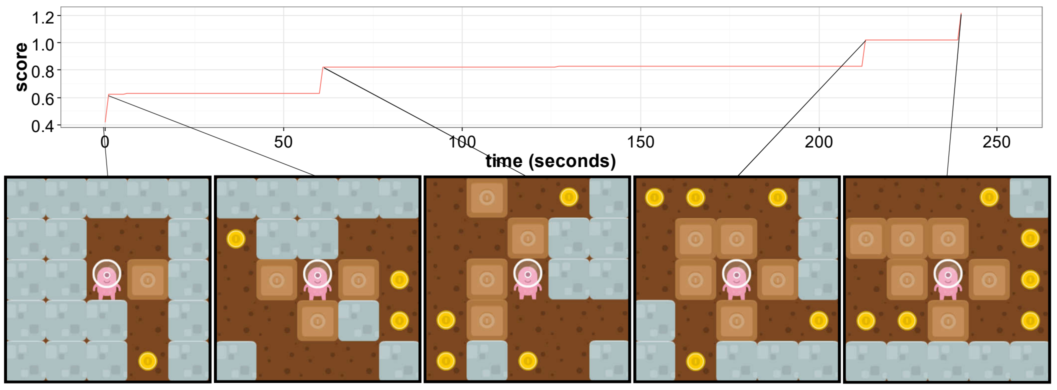


Figure 6.4: *Generating Level Sets.* (Top) The evolution of the best score from a single run of MCTS. (Bottom) Several levels generated from the same run. Later levels have higher score, and are therefore predicted to be more difficult.

Metrics which measured the placement of boxes and goals:

- *Box Count.* The number of boxes on the board.
- *Goal Distance.* The average distance between all possible pairings of boxes and goals

And metrics which measured how congested the paths from boxes to their goals were:

- *Congestion v1.* A weighted sum of the number of boxes, goals, and obstacles in the bounding rectangle between a box and its goal.
- *Congestion v2.* A refinement on the above congestion measure designed to maximize correlation with perceived difficulty (see below).

Importantly, each of these metrics can be computed in just a few microseconds, even for larger boards, allowing them to be efficiently used during MCTS rollout evaluation.

To test the efficacy of candidate features, the signed Pearson correlation coefficient r was computed for each feature with respect to perceived difficulty of the puzzles. For features which contained tuning parameters, we ran a grid search to find which parameters yielded the highest correlation. Table 6.1 shows the correlation between each metric and the perceived difficulties of the puzzles. We also show the correlation with only the procedural generated puzzles (PCG) tested, as the human crafted puzzles tended to have a significant effect on the analysis.

Looking at the correlations we can see several interesting trends. For example, the tile mixing metric is well correlated with difficulty for the entire dataset, but when only computer generated levels are considered the metric is not very predictive. In contrast, the simpler metric penalizing 3x3 blocks is more consistent. Likewise, the total distance the user must push all the boxes is slightly less correlated than the simpler approach of just counting the number of boxes to push (recall that boxes that are not pushed at least two spaces will be removed).

Simpler methods were not always the most predictive. In particular, the first version of the congestion metric was a weighted sum of the number of boxes b_i, number of goals g_i, and number of obstacles o_i within the bounding rectangle between the start and the goal for each box i. That is

$$\sum_{i=1}^{n} \alpha s_i + \beta g_i + \gamma o_i. \tag{6.1}$$

where, n is the number of boxes to be pushed, and α, β, and γ are scaling weights. While intuitive and relatively well correlated with difficulty for computer generated puzzles, this simple metric was almost completely uncorrelated with level difficulty when including human designed puzzles, even after tuning the values of α, β, and γ. Investigating the puzzles suggests this lack of correlation arises in part because the metric rewards pushing a box past obstacles even if there are no other boxes directly in the way. To address this issue, we refined the metric to be

$$\sum_{i=1}^{b} \frac{\alpha s_i + \beta g_i}{\gamma(A_i - o_i)}, \tag{6.2}$$

where A_i is the total area enclosed in the rectangle from the box i to its goal. The intent was to make the metric reward box paths that actually encounter boxes and obstacles, instead of just having them nearby an otherwise unconstrained path. While this was a small change to the measure of congestion, this new metric now correlates well with difficulty in both procedurally-generated and human-generated levels.

Feature	r value (PCG levels)	r value (all levels)
Tile Mixing	-0.05	0.17
3x3 Block	0.09	0.24
Box Count	0.48	0.23
Goal distance	-0.16	0.20
Congestion v1	0.41	0.05
Congestion v2	0.43	0.32

Table 6.1: Correlation (Pearson r correlation coefficients) for six features from the user study. The most correlated features were used for level evaluation function.

6.5.3 Level Evaluation

Using the results from our feature analysis, we developed an evaluation function for use in MCTS. For game playing AI, evaluation functions generally map to 0 for loss, 0.5 for a tie, and 1 for a win, with MCTS implementations typically calibrated to optimize on this scale. For Sokoban puzzle generation, this is not directly applicable (as the measure of success is not analogous to loss/tie/win). Instead, we propose to use a weighted combination of puzzle features to estimate the difficulty on a scale close to this 0 to 1 range.

By optimizing several metrics which are each independently correlated with puzzle difficulty, MCTS can be used to find more difficult puzzles than optimizing any one feature alone. Here, we used the *Box Count*, *3x3 Blocks*, and *Congestion v2*, as they were the most positively correlated with difficulty. Additionally, each of these metrics captures an intuitive aspect of what makes an interesting Sokoban level: the 3x3 Blocks metric (P_b) rewards heterogeneous landscapes, and discourages large clearings which are easy to navigate; the Congestion metric (P_c) rewards box paths which overlap with each other and are thereby likely to develop precedence constraints between box pushes; and the Box count (n) rewards levels with more boxes which makes complex interactions between boxes more likely. The resulting function is as follows:

$$f(P) = \frac{w_b P_b + w_c P_c + w_n n}{k} \tag{6.3}$$

The parameter k is employed to help normalize scores to the range of 0 to 1, though some of our top scoring puzzles can fall outside of this range.

While generating puzzles, $f(P)$ was used to evaluate MCTS rollouts. The weights w_c, w_b, and w_n were set empirically to be 1, 5, and 10 respectively and k was set to 50.

6.6 Generating Level Sets

A given run of the MCTS tree search will generate several levels of increasing (predicted) difficulty. We exploit this feature to reach our goal of creating a *level set*, that is, a series of levels that are of increasing difficulty. Because MCTS is an anytime algorithm that explores a wide, randomized section of the search space, it is well suited for this task; each run of MCTS creates several levels as it explores deeper in the tree, each with increasing difficulty.

While each run of MCTS can generate a large number of levels, many are slight variations of each other. To help create variation in the level sets, we chose a subset of these levels with different estimated difficulty scores. Figure 6.4 shows the results from one of these level sets. The entire run of MCTS for this set took 240s, and generated 20 of levels of varying difficulty.

6.7 Analysis and Discussion

Our approach efficiently generates dozens of levels with monotonically increasing scores within 5 minutes on on a laptop using a single core of an Intel i7 2.2 GHz processor with 8GB memory. We observe that it generates more levels with low and medium scores than high scores. An instance of this behavior can be seen in Figure 6.4.

To validate our updated evaluation function, we performed a second user-study on 20 levels generated in a single run (a subset of which is shown in Figure 6.4). This user study included 6 participants who provided 210 level comparisons. The perceived difficulty scores were then compared to the scores assigned by our evaluation function. The results can be seen in Figure 6.5. We observe a very high correlation ($r^2 = 0.91$) between the perceived difficulty of a level and the score assigned by MCTS. This confirms that MCTS will produce levels of increasing difficulty by optimizing this function.

Our method is capable of producing a wide variety of levels. Because MCTS is a

Score	Size (Empty Tiles)	Num. Boxes	Computation Time (s)
0.4	10.8	2.0	0.01
0.8	16.8	4.2	0.54
1.2	21.1	6.0	39.7
1.4	27.3	7.0	120

Table 6.2: **Puzzle Scaling.** Average computation time to find puzzles of various scores and size. While smaller puzzles with few boxes can be found in a under a second, scaling to larger sizes and box counts requires several minutes of computation. Results are averaged over 5 runs with with different random seeds.

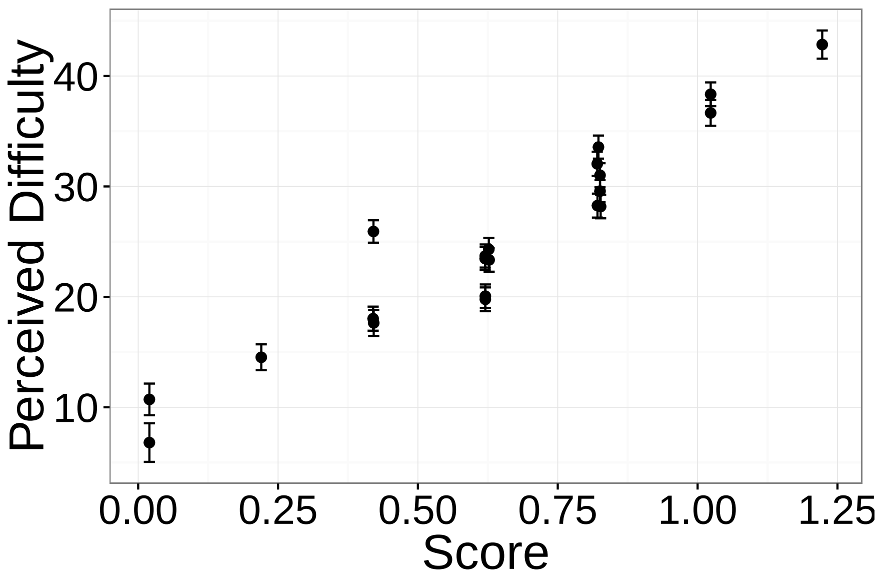


Figure 6.5: The evaluation score of the puzzles in the generated level set were well correlated with perceived difficulty (r^2 = .91).

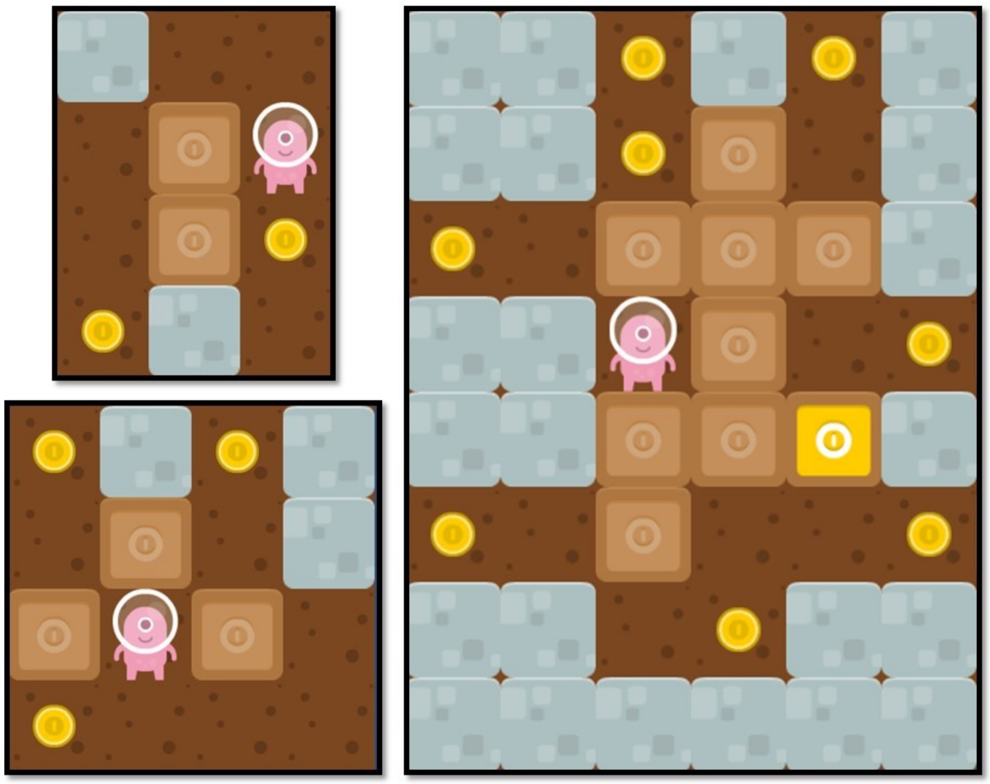


Figure 6.6: Procedurally generated puzzles of varying sizes

stochastic algorithm, each run naturally generates different levels from previous runs, and even within a single run (see Figure 6.4). Additional variation can be achieved by changing the maximum size of the board, randomizing the start position of the agent, or limiting the number of boxes in a level (see Figure 6.6). Figures 6.1, 6.2, 6.3, and 6.6 showcase the variety in the puzzles that are generated.

Limitations Generation of large puzzles remains a bottleneck as the time grows exponentially as the explored space and number of boxes grow linearly (see Table 6.2). This is due to a quickly growing branching factor in the puzzle initialization phase; every *Delete obstacle* action that is taken adds up to three more available *Delete obstacle* actions. Additionally, while our evaluation function is very highly correlated with perceived difficulty, the levels that our method generates are typically easier than human generated puzzles. Box count continues to be a main factor in both puzzle score and perceived difficulty. While increasing the number of boxes does increase puzzle complexity, many human designed puzzles tend to focus on creating complexity through other means.

6.8 Conclusions

In this work we have proposed and implemented a method for Sokoban puzzle generation. We formulated the problem as MCTS optimization, generating puzzles through simulated gameplay to ensure solvability. We developed an evaluation function with a data-driven approach, utilizing a user study to find puzzle features well correlated with perceived difficulty. Our method is efficient, producing a variety of puzzles with monotonically increasing scores within minutes. We validated our evaluation function through an additional user study and show that it correlates very well with perceived difficulty.

Going forward, we plan to investigate ways of efficiently creating larger puzzles of increasing difficulty. Some ways to overcome the current challenges in scaling up puzzles may include composing larger puzzles from smaller puzzle elements, reducing the size of the search space via data-driven heuristics, and exploring if some properties of optimal solutions may be computed quickly. Additionally, we plan to perform a follow up user study focused on larger levels and those of very high difficulty. Lastly, we plan to parallelize MCTS to increase computation performance.

Chapter 7

Conclusions and Future Work

In this dissertation, we addressed several challenges such as scalability, support for diverse action sets, and infinite policy generation which often arise in highly coordinated planning problems that have large search spaces. We have adapted Monte Carlo Tree Search algorithm for coordinated planning problems in both physical and abstract domains. We have shown that sampling-based stochastic tree search techniques can be adapted to efficiently find approximate solutions to large-scale planning problems that arise in a variety of domains including multi-robot planning and procedural content generation. These stochastic approaches can be successfully applied to domains with large state spaces, and their sampling-based nature allows flexibility in the details of the problem formulation and supports quick feedback for user-in-the-loop interaction.

7.1 Summary of Contributions

Our main contributions are as follows:

1. We propose Monte Carlo Tree Search with Useful Cycles algorithm [3, 150], i.e. an extended version of MCTS to search for infinite length multi-robot patrolling policies. Our approach improves the state of the art in the sense that it can be easily adapted to different intrusion models and arbitrary environments as long as an MDP simulator exists for the formulated problem.

2. We have proposed a novel parallelization technique for MCTS algorithm along with

employing branch and bound technique to improve search success for multi-robot task allocation problem with time windows and capacity constraints [4, 5]. Our approach is general unlike previous heuristics tailored for specific problems and it maintains completeness of MCTS. It finds near-optimal solutions for non-trivial problems in an existing test benchmarks within an hour.

3. We have proposed an MCTS based story generation algorithm [6, 7] along with a Bayesian inference method that can learn partial domain knowledge to be used by the planner. The main novelty of our approach improving the state-of-the art is its ability to allow the user to interact with the planner by changing believability of actions and interact with the planner in real-time.

4. We have proposed an MCTS based approach that creates Sokoban puzzles through simulated game play, guaranteeing solvability in all generated puzzles without checking for a solution [8, 9]. The concept of content generation with simulated play can be a touchstone and applied to other puzzles.

7.2 Limitations and Future Work

For MCTS, the main limitation is the further scalability for very large domains although MCTS is still one of the best options for planning domains with large branching factors. We can use pruning based heuristics to improve the scalability, but those heuristics render MCTS algorithm a locally optimal algorithm. To further improve scalability, one avenue is the search parallelization. However parallelization techniques are unlikely to provide more than linear improvement while problem complexity grows exponentially in the problem size. Recently, the authors in [151, 152] propose approaches that aggregate state space to obtain an abstracted state space so as to improve scalability of sampling based approaches. One avenue is to integrate these approaches to improve the planner performance for larger problems.

For multi-robot task allocation work, we did not consider possible traffic/contention during optimization. It is possible that theoretically optimal task allocation solutions causes traffic among robots and perform poorly on the field. We plan to adapt our planner so that it encourages separation between robots to prevent traffic and it generates flexible scheduling in order to complete tasks in a robust way in real world settings where delays and uncertainties are inevitable.

For multi-robot patrolling, our approach scales large environments only by using pruning techniques that result in locally optimal solutions. One possible way to overcome this challenge is to use an approach where the environment is initially coarsened enough to find a quick solution at a higher level, and then individual solutions are generated for each high level plan iteratively until finding primitive plans as described by the authors in [153] for efficient path-finding.

For Sokoban puzzle generation work, there are several avenues for future work. Currently, our method generates puzzles mostly in the form of dynamic topology [154] where the difficulty is mainly achieved through temporal order of box pushing as optimized by our metrics. The player needs to push a box to create an opening for other boxes. One direction would be understanding the features that would enable the planner to generate puzzles in the form of static topology as well. Future work also includes comparison with different sampling based approaches such as Nested Monte Carlo search proposed in [155].

References

[1] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, et al. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Bilal Kartal, Julio Godoy, Ioannis Karamouzas, and Stephen J Guy. Stochastic tree search with useful cycles for patrolling problems. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1289–1294, 2015.

[4] Bilal Kartal, Ernesto Nunes, Julio Godoy, and Maria Gini. Monte carlo tree search for multi-robot task allocation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] Bilal Kartal, Ernesto Nunes, Julio Godoy, and Maria Gini. Monte carlo tree search with branch and bound for multi-robot task allocation. In *The IJCAI-16 Workshop on Autonomous Mobile Service Robots*, 2016.

[6] Bilal Kartal, John Koenig, and Stephen J Guy. Generating believable stories in large domains. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[7] Bilal Kartal, John Koenig, and Stephen J Guy. User-driven narrative variation in large story domains using monte carlo tree search. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 69–76, 2014.

[8] Bilal Kartal, Nick Sohre, and Stephen Guy. Generating sokoban puzzle game levels with monte carlo tree search. In *The IJCAI-16 Workshop on General Game Playing*, 2016.

[9] Bilal Kartal, Nick Sohre, and Stephen Guy. Data-driven sokoban puzzle generation with monte carlo tree search. In *Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016.

[10] Frank Von Martial. *Coordinating plans of autonomous agents.* Springer-Verlag Berlin, 1992.

[11] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[12] Jur Van den Berg, Jack Snoeyink, Ming C Lin, and Dinesh Manocha. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and systems*, pages 2–3, 2009.

[13] Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard Segal. Fuegoan open-source framework for board games and go engine based on monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):259–270, 2010.

[14] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices.* John Wiley & Sons, 2011.

[15] SaiDhiraj Amuru, Cem Tekin, Mihaela van der Schaar, and R Michael Buehrer. A systematic learning method for optimal jamming. In *Communications (ICC), 2015 IEEE International Conference on*, pages 2822–2827. IEEE, 2015.

[16] Shuvra S Bhattacharyya, Mihaela van der Schaar, Onur Atan, Cem Tekin, and Kishan Sudusinghe. Data-driven stream mining systems for computer vision. In *Advances in Embedded Computer Vision*, pages 249–264. Springer, 2014.

[17] Cem Tekin, Onur Atan, and Mihaela Van Der Schaar. Discover the expert: Context-adaptive expert selection for medical diagnosis. *Emerging Topics in Computing, IEEE Transactions on*, 3(2):220–234, 2015.

[18] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[19] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.

[20] Radha-Krishna Balla and Alan Fern. Uct for tactical assault planning in real-time strategy games. In *IJCAI*, pages 40–45, 2009.

[21] Niek GP Den Teuling and Mark HM Winands. Monte-carlo tree search for the simultaneous move game tron. *Univ. Maastricht, Netherlands, Tech. Rep*, 2011.

[22] Tom Pepels, Mark HM Winands, and Marc Lanctot. Real-time monte carlo tree search in ms pac-man. *IEEE Trans. on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.

[23] Marc Lanctot, Mark HM Winands, Tom Pepels, and Nathan R Sturtevant. Monte Carlo tree search with heuristic evaluations using implicit minimax backups. In *Proc. IEEE Conf. on Computational Intelligence and Games (CIG)*, pages 1–8, 2014.

[24] Frederik Frydenberg, Kasper R Andersen, Sebastian Risi, and Julian Togelius. Investigating MCTS modifications in general video game playing. In *Proc. IEEE Conf. on Computational Intelligence and Games (CIG)*, pages 107–113, 2015.

[25] Viliam Lisỳ, Marc Lanctot, and Michael Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent*

Systems, pages 27–36. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[26] Erik Steinmetz and Maria Gini. Mining expert play to guide monte carlo search in the opening moves of go. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 801–807. AAAI Press, 2015.

[27] Johannes Heinrich and David Silver. Smooth uct search in computer poker. *Proceedings of the 24th International Joint Conference on Artifical Intelligence*, 2015.

[28] Alberto Uriarte and Santiago Ontañón. High-level representations for game-tree search in rts games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[29] Edward J Powley, Daniel Whitehouse, Peter Cowling, et al. Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 234–241. IEEE, 2012.

[30] Stefan Edelkamp and Max Gath. Solving single vehicle pickup and delivery problems with time windows and capacity constraints using nested monte-carlo search. *ICAART (1)*, pages 22–33, 2014.

[31] Mohamed Amer, Sinisa Todorovic, Alan Fern, and Song-Chun Zhu. Monte carlo tree search for scheduling activity recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1353–1360, 2013.

[32] Mikko Lauri, Nikolay Atanasov, George Pappas, and Risto Ritala. Active Object Recognition via Monte Carlo Tree Search. In *ICRA Workshop on Beyond Geometric Constraints*, 2015.

[33] Adam James Summerville, Shweta Philip, and Michael Mateas. Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[34] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[35] Nick Sephton, Peter I Cowling, and Nicholas H Slaven. An experimental study of action selection mechanisms to create an entertaining opponent. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 122–129. IEEE, 2015.

[36] Steven James, Benjamin Rosman, and George Konidaris. An investigation into the effectiveness of heavy rollouts in uct. In *The IJCAI-16 Workshop on General Game Playing*, page 55, 2016.

[37] Abdallah Saffidine, Tristan Cazenave, and Jean Méhat. Ucd: Upper confidence bound for rooted directed acyclic graphs. *Knowledge-Based Systems*, 34:26–33, 2012.

[38] M Sakti Alvissalim, Big Zaman, Z Ahmad Hafizh, M Anwar Ma'sum, Grafika Jati, Wisnu Jatmiko, and Petrus Mursanto. Swarm quadrotor robots for telecommunication network coverage area expansion in disaster area. In *SICE Annual Conference (SICE), 2012 Proceedings of*, pages 2256–2261. IEEE, 2012.

[39] Jnaneshwar Das, Gareth Cross, Chao Qu, Anurag Makineni, Pratap Tokekar, Yash Mulgaonkar, and Vijay Kumar. Devices, systems, and methods for automated monitoring enabling precision agriculture. In *IEEE International Conference on Automation Science and Engineering (CASE), vol., no*, pages 462–469, 2015.

[40] Volkan Isler, Narges Noori, Patrick Plonski, Alessandro Renzaglia, Pratap Tokekar, and Josh Vander Hook. Finding and tracking targets in the wild: Algorithms and field deployments. In *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–8. IEEE, 2015.

[41] Haluk Bayram, Joshua Vander Hook, and Volkan Isler. Gathering bearing data for target localization. *IEEE Robotics and Automation Letters*, 1(1):369–374, 2016.

[42] Bobby Davis, Ioannis Karamouzas, and Stephen J Guy. C-opt: Coverage-aware trajectory optimization under uncertainty. *IEEE Robotics and Automation Letters*, 1(2):1020–1027, 2016.

[43] Mikko Lauri, Aino Ropponen, and Risto Ritala. Risk-aversive optimal planning of sensing. In *Journal of Physics: Conference Series*, volume 588, pages 12023–12028, 2015.

[44] Pratap Tokekar and Vijay Kumar. Visibility-based persistent monitoring with robot teams. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3387–3394. IEEE, 2015.

[45] J Melin, M Lauri, A Kolu, J Koljonen, and R Ritala. Cooperative sensing and path planning in a multi-vehicle environment. *IFAC-PapersOnLine*, 48(9):198–203, 2015.

[46] David Salda, Reza Javanmard Alitappeh, Luciano CA Pimenta, Renato Assun, Mario FM Campos, et al. Dynamic perimeter surveillance with a team of robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5289–5294. IEEE, 2016.

[47] Mazda Ahmadi and Peter Stone. A multi-robot system for continuous area sweeping tasks. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1724–1729. IEEE, 2006.

[48] Yann Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings of Intelligent Agent Technology (IAT)*, 2004.

[49] Pooyan Fazli, Alireza Davoodi, Philippe Pasquier, and Alan K Mackworth. Complete and robust cooperative robot area coverage with limited range. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5577–5582. IEEE, 2010.

[50] Pooyan Fazli and Alan K Mackworth. The effects of communication and visual range on multi-robot repeated boundary coverage. In *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, pages 1–8. IEEE, 2012.

[51] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Multi-Agent-Based Simulation II*, pages 155–170. Springer, 2003.

[52] Fabio Pasqualetti, Antonio Franchi, and Francesco Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *Robotics, IEEE Transactions on*, 28(3):592–606, 2012.

[53] David Portugal, Charles Pippin, Rui P Rocha, and Henrik Christensen. Finding optimal routes for multi-robot patrolling in generic graphs. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 363–369. IEEE, 2014.

[54] Nicola Basilico, Nicola Gatti, and Federico Villa. Asynchronous multi-robot patrolling against intrusions in arbitrary topologies. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[55] Alessandro Marino, Lynne Parker, Gianluca Antonelli, and Fabrizio Caccavale. Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In *IEEE International Conference on Robotics and Automation*, pages 831–836, 2009.

[56] Tiago Sak, Jacques Wainer, and Siome Klein Goldenstein. Probabilistic multiagent patrolling. In *Advances in Artificial Intelligence-SBIA 2008*, pages 124–133. Springer, 2008.

[57] Noa Agmon, Gal A Kaminka, and Sarit Kraus. Multi-robot adversarial patrolling: Facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011.

[58] Ahmad Bilal Asghar and Stephen L Smith. Stochastic patrolling in adversarial settings. In *American Control Conference*. IEEE, 2016.

[59] Efrat Sless, Noa Agmon, and Sarit Kraus. Multi-robot adversarial patrolling: facing coordinated attacks. In *Autonomous agents and multi-agent systems*, pages 1093–1100, 2014.

[60] Armando Marino, Gianluca Antonelli, A Pedro Aguiar, António Pascoal, and Stefano Chiaverini. A decentralized strategy for multirobot sampling/patrolling: Theory and experiments. *Control Systems Technology, IEEE Transactions on*, 23(1):313–322, 2015.

[61] Tauhidul Alam, Matthew Edwards, Leonardo Bobadilla, and Dylan Shell. Distributed multi-robot area patrolling in adversarial environments. In *International Workshop on Robotic Sensor Networks, Seattle, WA, USA*, 2015.

[62] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Distributed on-line dynamic task assignment for multi-robot patrolling. *Autonomous Robots*, pages 1–25, 2016.

[63] Derya Aksaray, Kevin Leahy, and Calin Belta. Distributed multi-agent persistent surveillance under temporal logic constraints. *IFAC-PapersOnLine*, 48(22):174–179, 2015.

[64] Elizabeth Jensen, Michael Franklin, Sara Lahr, and Maria Gini. Sustainable multi-robot patrol of an open polyline. In *IEEE International Conference on Robotics and Automation*, pages 4792–4797, 2011.

[65] Praveen Paruchuri, Jonathan P Pearce, Milind Tambe, Fernando Ordonez, and Sarit Kraus. An efficient heuristic approach for security against multiple adversaries. In *Autonomous agents and multiagent systems*, page 181, 2007.

[66] Charles Pippin and Henrik Christensen. Trust modeling in multi-robot patrolling. In *IEEE International Conference on Robotics and Automation*, pages 59–66, 2014.

[67] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Minimizing communication latency in multirobot situation-aware patrolling. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 616–622. IEEE, 2015.

[68] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. In *Technological Innovation for Sustainability*, pages 139–146. Springer, 2011.

[69] Dennis Nieuwenhuisen and Mark H Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 446–452, 2004.

[70] Narges Noori, Alessandro Renzaglia, and Volkan Isler. Searching for a one-dimensional random walker: Deterministic strategies with a time budget when

crossing is allowed. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4811–4816, 2013.

[71] Turtlebot 2. `http://www.turtlebot.com`, 2016.

[72] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5, 2009.

[73] Esha D Nerurkar and Stergios Roumeliotis. Power-slam: a linear-complexity, anytime algorithm for slam. *The International Journal of Robotics Research*, page 0278364910390539, 2011.

[74] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fast-slam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002.

[75] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46, 2007.

[76] Joao Machado Santos, David Portugal, and Rui P Rocha. An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 2013.

[77] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[78] Lantao Liu and Dylan A Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33(3):291–307, 2012.

[79] Julio Godoy and Maria Gini. Task allocation for spatially and temporally distributed tasks. In *Proc. Int'l Conf. on Intelligent Autonomous Systems*, pages 603–612, 2012.

[80] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Robotics: Science and Systems*, 2013.

[81] S. S. Ponda, J. Redding, Han-Lim Choi, J.P. How, M. Vavrina, and J. Vian. Decentralized planning for complex missions with dynamic communication constraints. In *Proc. American Control Conf.*, pages 3998–4003, 2010.

[82] Ernesto Nunes and Maria Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[83] Mitchell McIntire, Ernesto Nunes, and Maria Gini. Iterated multi-robot auctions for precedence-constrained task scheduling. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1078–1086. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[84] Haluk Bayram and H Işıl Bozma. Coalition formation games for dynamic multirobot tasks. *The International Journal of Robotics Research*, 35(5):514–527, 2016.

[85] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[86] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple UAVs with timing constraints and loitering. In *Proc. American Control Conf.*, pages 5311–5316, June 2003.

[87] Hakim Mitiche, Dalila Boughaci, and Maria Gini. Efficient heuristics for a time-extended multi-robot task allocation problem. In *New Technologies of Information and Communication (NTIC), 2015 First International Conference on*, pages 1–6. IEEE, 2015.

[88] Mary Koes, Illah R. Nourbakhsh, and Katia P. Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1292–1297, 2005.

[89] Changjoo Nam and Dylan Shell. Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Trans. on Automation Science and Engineering*, 12(3):889–900, 2015.

[90] G. Ayorkor Korsah, Anthony Stentz, M. Bernardine Dias, and Imran Aslam Fanaswala. Optimal vehicle routing and scheduling with precedence constraints and location choice. In *Workshop on Intelligent Autonomous Systems at IEEE ICRA*, 2010.

[91] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, February 2005.

[92] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139, February 2005.

[93] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel Monte-Carlo Tree Search. In *Computers and Games*, pages 60–71. Springer, 2008.

[94] Alan Fern and Paul Lewis. Ensemble monte-carlo planning: An empirical study. In *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[95] Marius M. Solomon. Vrptw benchmark problems. `http://web.cba.neu.edu/~msolomon/problems.htm`, 2005.

[96] Markus Eger, Camille Barot, and R Michael Young. Impulse: A formal characterization of story. In *OASIcs-OpenAccess Series in Informatics*, volume 45. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[97] Stephen G Ware, Robert M Young, Brent Harrison, and David L Roberts. A computational model of plan-based narrative conflict at the fabula level. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):271–288, 2014.

[98] Nicholas Davis, Margeaux Comerford, Mikhail Jacob, Chih-Pin Hsiao, and Brian Magerko. An enactive characterization of pretend play. In *Proceedings of the 2015*

ACM SIGCHI Conference on Creativity and Cognition, pages 275–284. ACM, 2015.

[99] Maria Fox and Derek Long. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124, 2003.

[100] Theo Wadsley and Malcolm Ryan. A belief-desire-intention model for narrative generation. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[101] Hans Brinke, Jeroen Linssen, and Mariët Theune. Hide and sneak: story generation with characters that perceive and assume. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[102] Mark O Riedl and Robert Michael Young. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39(1):217–268, 2010.

[103] Mei Si, Stacy C Marsella, and David V Pynadath. Thespian: Using multi-agent fitting to craft interactive drama. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 21–28, 2005.

[104] David L Roberts and Charles L Isbell. Lessons on using computationally generated influence for shaping narrative experiences. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):188–202, 2014.

[105] Margaret Sarlej and Malcolm Ryan. Generating stories with morals. In *Interactive Storytelling*, pages 217–222. Springer, 2013.

[106] Barbaros Bostan and Tim Marsh. The interactiveof interactive storytelling: customizing the gaming experience. In *Entertainment Computing-ICEC 2010*, pages 472–475. Springer, 2010.

[107] Mirjam P Eladhari, Philip L Lopes, and Georgios N Yannakakis. Interweaving story coherence and player creativity through story-making games. In *Interactive Storytelling*, pages 73–80. Springer, 2014.

[108] Reid Swanson and Andrew S Gordon. Say anything: Using textual case-based reasoning to enable open-domain interactive storytelling. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(3):16, 2012.

[109] Pablo Gervás, Belén Díaz-Agudo, Federico Peinado, and Raquel Hervás. Story plot generation based on cbr. *Knowledge-Based Systems*, 18(4):235–242, 2005.

[110] Mubbasir Kapadia, Jessica Falk, Fabio Zünd, Marcel Marti, Robert W Sumner, and Markus Gross. Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, pages 85–92, 2015.

[111] Mariet Theune, Sander Faas, Er Faas, Anton Nijholt, and Dirk Heylen. The virtual storyteller: Story creation by intelligent agents. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, pages 204–215, 2003.

[112] Michael Brenner. Creating dynamic story plots with continual multiagent planning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2010.

[113] Jonathan Teutenberg and Julie Porteous. Efficient intent-based narrative generation using multiple planning agents. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 603–610, 2013.

[114] Seung Y Lee, Jonathan P Rowe, Bradford W Mott, and James C Lester. A supervised learning framework for modeling director agent strategies in educational interactive narrative. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):203–215, 2014.

[115] Mark J Nelson, David L Roberts, Charles L Isbell Jr, and Michael Mateas. Reinforcement learning for declarative optimization-based drama management. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 775–782, 2006.

[116] Hong Yu and Mark Owen Riedl. Personalized interactive narratives via sequential recommendation of plot points. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):174–187, 2014.

[117] Julie Porteous, Alan Lindsay, Jonathon Read, Mark Truran, and Marc Cavazza. Automated extension of narrative planning domains with antonymic operators. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1547–1555, 2015.

[118] Alan Lindsay, Fred Charles, Jonathon Read, Julie Porteous, Marc Cavazza, and Gersende Georg. Generation of non-compliant behaviour in virtual medical narratives. In *Intelligent Virtual Agents*, pages 216–228. Springer International Publishing, 2015.

[119] Joshua McCoy, Mike Treanor, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. Comme il faut: A system for authoring playable social models. In *AIIDE*, 2011.

[120] Boyang Li, Stephen Lee-Urban, George Johnston, and Mark Riedl. Story generation with crowdsourced plot graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[121] Josep Valls-Vargas, Jichen Zhu, and Santiago Ontañón. Toward automatic role identification in unannotated folk tales. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[122] Jonathan Rowe, Bradford Mott, and James Lester. Optimizing player experience in interactive narrative planning: A modular reinforcement learning approach. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[123] Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying mcts for human-like general video game playing. *Proceedings of the 25th International Joint Conference on Artifical Intelligence*, 2016.

[124] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. " O'Reilly Media, Inc.", 2009.

[125] Hugo Liu and Push Singh. Conceptnet a practical commonsense reasoning toolkit. *BT technology journal*, 22(4):211–226, 2004.

[126] Walaa Baghdadi, Fawzya Shams Eddin, Rawan Al-Omari, Zeina Alhalawani, Mohammad Shaker, and Noor Shaker. A procedural method for automatic generation of spelunky levels. In *European Conference on the Applications of Evolutionary Computation*, pages 305–317. Springer, 2015.

[127] Sam Snodgrass and Santiago Ontañón. Generating maps using markov chains. In *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence and Game Aesthetics*, pages 25–28, 2013.

[128] Sam Snodgrass and Santiago Ontañón. A hierarchical mdmc approach to 2d video game map generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[129] Sam Snodgrass and Santiago Ontañón. Controllable procedural content generation via constrained multi-dimensional markov chain sampling. *Proceedings of the 25th International Joint Conference on Artifical Intelligence*, 2016.

[130] David Maung and Roger Crawfis. Applying formal picture languages to procedural content generation. In *Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES), 2015*, pages 58–64, 2015.

[131] N Sturtevant. An argument for large-scale breadthfirst search for game design and content generation via a case study of fling. In *AI in the Game Design Process (AIIDE workshop)*, 2013.

[132] Umair Z Ahmed, Krishnendu Chatterjee, and Sumit Gulwani. Automatic generation of alternative starting positions for simple traditional board games. In *Twenty-Ninth AAAI Conf. on Artificial Intelligence*, 2015.

[133] David Stammer, Tobias Gunther, and Mike Preuss. Player-adaptive spelunky level generation. In *Computational Intelligence and Games, 2015 IEEE Conference on*, pages 130–137, 2015.

[134] Adam M Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187–200, 2011.

[135] Matthew Guzdial and Mark O Riedl. Toward game level generation from gameplay videos. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2015.

[136] Aaron Isaksen and Andy Nealen. Comparing player skill, game variants, and learning rates using survival analysis. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[137] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, 2011.

[138] Joseph Culberson. Sokoban is PSPACE-complete. In *Proceedings in Informatics*, volume 4, pages 65–76, 1999.

[139] Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1):219–251, 2001.

[140] André G Pereira, Marcus Ritt, and Luciana S Buriol. Optimal sokoban solving using pattern databases with specific domain knowledge. *Artificial Intelligence*, 227:52–70, 2015.

[141] Diego Perez, Spyridon Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary mcts for general video game playing. In *Proc. IEEE Conf. on Computational Intelligence and Games (CIG)*, pages 1–8, 2014.

[142] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. Automatic making of sokoban problems. In *Pacific Rim International Conference on Artificial Intelligence*, pages 592–600. Springer, 1996.

[143] Joshua Taylor and Ian Parberry. Procedural generation of sokoban levels. In *North American Conf. on Intelligent Games and Simulation*, pages 5–12, 2011.

[144] Marc Van Kreveld, Maarten Loffler, and Paul Mutser. Automated puzzle difficulty estimation. In *Proc. IEEE Conf. on Computational Intelligence and Games (CIG)*, pages 415–422, 2015.

[145] Petr Jarušek and Radek Pelánek. Difficulty rating of sokoban puzzle. In *Stairs*, volume 222, page 140, 2010.

[146] Daniel Ashlock and Justin Schonfeld. Evolution for automatic assessment of the difficulty of sokoban boards. In *Evolutionary Computation*, pages 1–8, 2010.

[147] Joshua Taylor, Thomas D Parsons, and Ian Parberry. Comparing player attention on procedurally generated vs. hand crafted sokoban levels with an auditory stroop test. In *Conf. on the Foundations of Digital Games*, 2015.

[148] David W. Skinner. Sokoban puzzle dataset microban. `http://www.abelmartin.com/rj/sokobanJS/Skinner/David%20W.%20Skinner%20-%20Sokoban.htm`, 2000.

[149] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill: A bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576, 2006.

[150] Bilal Kartal. Monte carlo tree search with useful cycles for motion planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA) PhD Forum*. IEEE, 2015.

[151] Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2446–2452, 2014.

[152] Hao Cui, Roni Khardon, Alan Fern, and Prasad Tadepalli. Factored mcts for large scale stochastic planning. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 3261–3267, 2015.

[153] Nathan Sturtevant and Michael Buro. Partial pathfinding using map abstraction and refinement. In *Proc. AAAI Conf. on Artificial Intelligence*, volume 5, pages 1392–1397, 2005.

[154] David Holland. The two styles of sokoban. `http://www.games4brains.de/twostyles.html`, 2001.

[155] Tristan Cazenave. Nested monte-carlo search. In *Proc. Int'l Joint Conf. on Artificial intelligence*, volume 9, pages 456–461, 2009.