ACTION SPACE REPRESENTATION IN COMBINATORIAL MULTI-ARMED BANDITS

Gijs-Jan Roelofs

Master Thesis 15-24

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Faculty of Humanities and Sciences
of Maastricht University

Thesis committee:

Dr. M.H.M. Winands
Dr. G.M. Schoenmakers

Maastricht University
Department of Knowledge Engineering
Maastricht, The Netherlands
July 2015

Preface

This master thesis is written in partial fulfillment of the requirements for the degree of Master of Science of Artificial Intelligence at the Department of Knowledge Engineering. This thesis is placed within the field of search techniques in computer games, specifically Monte-Carlo Tree Search and Combinatorial Multi-Armed Bandit problems. I would like to thank my thesis supervisor Dr. Mark Winands, for his insight and above all tolerance for endless discussion, along with the whole DKE staff for my formative years at DKE. I would also like to thank Dries de Rydt, Taghi Aliyev, and other friends who endured my endless talks on the subjects within this thesis. Finally but certainly not least, I would like to thank my family for their unconditional support during my education at DKE.

Summary

Monte-Carlo sampling based approaches have been applied to great success in domains of online planning with sizable state spaces. When applied to an combinatorial action space state-of-the-art algorithms such as Monte-Carlo Tree Search (MCTS) fail to provide similar levels of performance as found when applied to a polynomial action space. Algorithms which do not approach the action space as enumerable, such as Naïve Monte-Carlo Sampling (NMC) and Linear Side Information (LSI), have had better success. In this thesis we experimentally analyze robustness of various algorithms in two domains with combinatorial action space and propose two augmentations to MCTS, Hierarchical Expansion (HE) and Dimensional Expansion (DE), to cope with an effectively innumerable action space. We then explore the implications of partial moves returned by the tree and propose an augmentation to MCTS for this problem based on RAVE.

The first problem domain is a simple model of a *Combinatorial Multi-Armed Bandit* (CMAB) problem. In this problem the player is tasked with choosing the optimal combination of arms for all bandits. The model is setup to perform a sensitivity analysis on the various algorithms with regards to the characteristics of the CMAB problem.

The second problem domain is Berlin, an online planning domain in the form of a deterministic, simultaneous turn-based strategy game with an combinatorial action space. The action space is represented as a decision making problem for a team of cooperative agents.

Both Dimensional Expansion (DE) and Hierarchical Expansion (HE) are shown to outperform NMC and LSI within the CMAB domain, while Hierarchical Expansion is shown to outperform DE, NMC, LSI and heuristic based approaches within the domain of Berlin. The partial move augmentations are shown to improve performance in both DE and HE.

Contents

1

Introduction

"Planning is the art and practice of thinking before acting."

– Patrik Haslum

Chapter overview: *The following chapter provides an introduction to this thesis consisting of the motivation for, and the techniques and problem domains explored within. The chapter closes with the goal, divided up into the problem statement and research questions, and a general outline of the structure of this thesis.*

1.1 Games & AI

Since the inception of the field of Artificial Intelligence (AI), decision-making has been one of its major sub fields. Within decision-making one of the most common problems is that of a single agent tasked with finding the optimal action, or sequence of actions, which achieves a given goal. Many real-world domains which derive from this problem are often too complicated to use in research, or result in experiments where the variables affecting the results cannot easily be separated. In these cases abstract models are often used to distill domains to their core problems which then in turn can be analyzed and reasoned about.

Games provide a step between abstract models and complex real-world problems. They allow specific problems to be explored without being exposed to the complexities of real-world constraints, but still complex enough to provide interesting problems. Furthermore, their ability to directly measure the performance of a found solution against a human player has always been a driving factor in research.

Chess has led to the discovery of many different techniques, and is an excellent example of this. What began with Alan Turing in 1953 [52] culminated in a master-level human player being defeated in 1997 by Deep Blue [10], and led to the discovery of such techniques as $\alpha\beta$-search [28], NegaScout [42] and Proof Number search [2].

1.1.1 Search Techniques

These techniques are based on a state-space based search, with the area of interest constrained to that considered most relevant for the agent's decision at hand. This reduction is done to deal with the so-called "curse of state dimensionality" [31]: the concept that the state-space grows too vast to approach as a whole with the, often limited, resources at hand.

One of the core assumptions in these techniques is that the outcome of an action or state can be predicted through the use of an evaluation function: a function which describes the current and future worth of a state for a player. The evaluation function is what enables the above outlined techniques to constrain the search to interesting states by pruning those states which are deemed uninteresting.

As the field of AI grew, games of ever-growing complexity were explored. Games such as Go prohibited the application of hitherto working techniques because of two reasons: first, the base assumption - the existence of an easily definable evaluation function - was invalidated and second, the branching factor

grew to such heights that even with heavy pruning the state-space could no longer be searched to any sufficient degree. Different approaches and models needed to be employed which did not rely on knowledge of the domain itself or exploration of the full state-space.

1.1.2 Dealing with State-Space

One such model is the Multi-Armed Bandit (MAB) [54]. MAB models the state as a sequence of slot machines, each when played providing a value drawn from a random distribution specific to that machine. The agent is represented as a gambler who, at each round, has to decide on which machine to play in an effort to maximize his reward.

The problem the gambler faces each round is that of making the choice between choosing a different machine in hope to gain more information on its expected payoff (*exploration*), and selecting the machine with the highest expected payoff thus far (*exploitation*). The basis of this problem is that we assume no knowledge on the problem domain at hand. Any knowledge gained, is gained through observation of the outcome of any executed actions.

The MAB is therefore an example of a model that can be used to represent many types of problems, essentially any problem in which we want to explore the consequences of a set of choices for which we do not yet know the outcome or payoff. By researching the implications of this model we gain knowledge on how to develop and apply techniques which can be used in a broad range of situations.

The application of MAB on the domain of Go led to the development of Monte-Carlo Tree Search (MCTS) [29, 16] and Upper Confidence Bounds (UCB) [16], and through them the rise of expert-level play for small (9×9) boards [22, 43].

The basis of MCTS is that the state-space is sampled to determine the value or expected reward of an action, requiring no knowledge on the domain through heuristic evaluation. The use of a selection policy to explore the tree ensures that the technique is able to handle domains with ever growing state complexity, outperforming previous algorithms [35].

The success in the domain of Go sparked interest in the application of the technique in different domains [8]. New levels of expert play in games were reached as MCTS was applied to new domains ranging from deterministic board games such as Hex [3], Havannah [18], real-time video games such as Ms. Pacman [21] and Total War: Rome II [26], and even non-deterministic domains such as Settlers of Catan [49] and Poker [41].

New domains were created specifically to expose the shortcomings of the techniques explored thus far. Arimaa [48] is one such domain, a turn-based strategy game in which an action is subdivided into one to four sub-actions. This subdivision leads to an action-space complexity which is combinatorial in nature and proved to be a challenging new problem. Where in Chess the average branching factor is 35, and in Go it increases to 200, in Arimaa this number reaches 17,000 on average and up to 50,000 in complicated mid-game situations [32]. The techniques used thus far can no longer be applied as they operate under the assumption that the action-space can be enumerated efficiently, or better said, that each action can be sampled at least once [38].

1.1.3 Dealing with Action-Space

This combinatorial action-space complexity introduces the concept of the "curse of action dimensionality" [38]: the concept of an action split up into multiple dimensions, each for which a decision needs to be made, is inherent to multi-agent strategy games.

In this type of game the optimal sequence of actions needs to be decided for a team of agents, represented by the units the player controls. This in turn results in a multi-dimensional, or combinatorial action space, as opposed to the single dimensional one traditionally found when dealing with planning for a single agent.

The complexity of this action space stems from its definition as a cross-product of all the action spaces of the individual agents. In both the domains of Arimaa [19, 32, 55] and StarCraft [47] a naive restructuring of the action space lead to increased performance. In this restructuring the action space was restructured such that each ply explored the variations of a part of the move. However, the implications of this restructuring, and variations thereof have not yet been explored. Within MCTS, the complexities of these action spaces have mostly been dealt with through abstracting over the domain [6, 15]. One approach was to use heuristics on the individual agents and let the search decide on the overall strategy of

aggregated groups of agents, named squads [47]. However, this only partly solved the issue, as it reduced the action space per dimension but did not address the combinatorial nature of the action space. A restructuring of the action space was still required in which each ply the variations of actions per squad were explored, instead of all possible actions of the player as a whole [47].

Apart from restructuring the action space, success was had by approaching these problems as Combinatorial Multi Armed Bandit problems (CMAB) [20] [13] [38]. Naive Monte-Carlo [38] (NMC) and Linear Side-Information [31] (LSI) algorithms are two algorithms which follow this approach and have had success over MCTS in the domain of μRTS.

The basis for both approaches is a relaxation of the problem. In the relaxed problem we operate under the assumption that the actions of an agent do not affect the outcome of the actions of an agent in the same team. In other words, the reward of an action is linear over its sub-actions.

Each technique then approaches a divide-and-conquer approach in which each dimension, or sub-action, is explored individually before the results of which are combined into an action.

In this thesis we explore the implications of various representations of the action space in the game tree and establish a formal definition which can be used to deal with these type of problems. We then compare the performance of MCTS with these extensions to that of the alternative proposed techniques, namely NMC and LSI, within the domain of Berlin, and in the abstract domain of Combinatorial Multi-Armed Bandits.

1.2 Problem Domains

In this thesis we experiment in two different domains: The online AI tournament of Berlin, and a domain devised for this thesis which is as close to the formal definition of a CMAB as possible.

1.2.1 Berlin

Berlin is a turn-based strategy game implemented for the Berlin AI competition, started in 2013 by the Thirdside Collective [51]. It is a strategy game in which the player focuses purely on the strategy and deployment of units. Each turn each player needs to make a decision simultaneously on how units move along the graph. All units are equal in strength, and the outcome of a battle is simply the remainder of units left on a node and thus deterministic. By abstracting over the tactical aspect, or combat, of strategy games the game allows experimentation to focus on the strategical decision making.

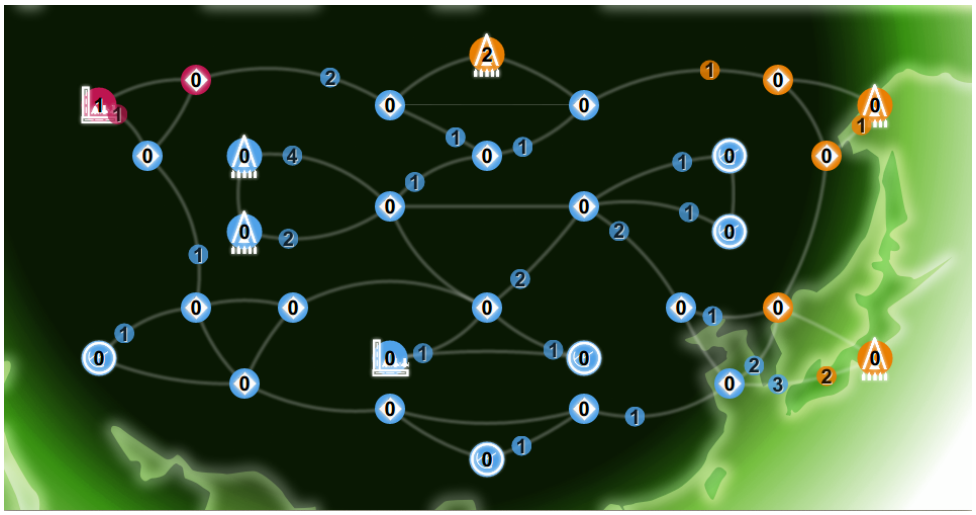

Figure 1.1: An example visualization of Berlin on http://berlin-ai.com/.

The complexity from Berlin rises from two aspects: the necessity to decide for each soldier, each turn, and the observation that the choice for each soldiers influences the outcome of choices made for other soldiers. The problem cannot be modeled as a MAB without the actions space expanding rapidly through its combinatorial nature, and this complexity is explored in Section 2.2.

1.2.2 Combinatorial Multi-Armed Bandit

The second domain proposed in this thesis is a straightforward implementation of the Combinatorial Multi-Armed Bandit (CMAB) [13]. In the CMAB model each round the player needs to decide between *super arms*, sets of arms, to be played instead of single arms. In other words we identify actions which can be broken up into multiple dimensions, each with their own set of sub-actions. In the domain of Berlin, a super arm could represent a decision for all soldiers in a single turn. The underlying arms could represent the choices per soldier. If the classical MAB framework were to be applied to Berlin, each of these super arms would need to be treated as an individual arm. This naive approach, in which we explore all actions per player per ply, suffers from two issues [13]:

- The action space may become exponential due to the combinatorial explosion.
- Any information, called the side-information [38], which could be observed on the underlying arms of a super-arm is discarded.

Side-information is the concept of learning the underlying worth of an underlying arm through its repeated use over multiple super-arms. The side-information is tightly linked to the linearity of the underlying value function in the domain. If a value function is linear in nature, then we can optimize the exploration of the combined action by determining the value of its sub-actions and optimizing for those. As the linearity of the value function decreases, and the value of arms becomes more and more intertwined, the correlation of the value of a single arm to the estimated worth of a super-arm becomes less significant.

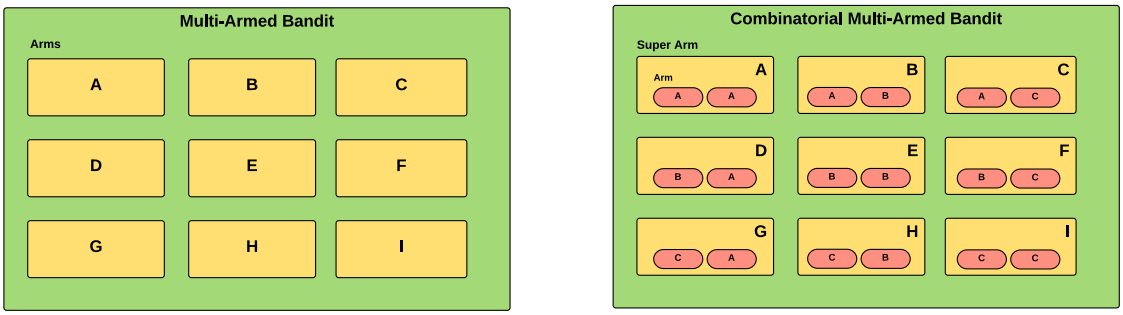

Figure 1.2: Action division between Multi-Armed Bandit and Combinatorial Multi-Armed Bandit.

Two techniques, LSI and NMC [31, 38], which have been proposed that simultaneously deal with the expansive action space and try to exploit this underlying side-information. However, so far these techniques have been tested in a single CMAB domain, μRTS, while the CMAB model introduces several variables that influence the complexity of the problem, namely:

- The number of bandits: the number of dimensions.
- The number of arms: the size of the action set of each dimension.
- The linear versus non-linear nature of the reward function, or interdependence between the reward functions of the individual dimensions.

By constructing a domain that is as close as possible to the model of a CMAB, we can explore how various algorithms respond to the variables in this model.

For consistency sake, in this thesis the *super arm* is referred to as the *combined action* cA, with its individual arms i, each with a respective set of sub-actions A_i. A single action from arm i, or set A_i, is denoted as a_i. The set of all combined actions possible from a given state is denoted as C. The details of the proposed domain, and its implications are further explored in Section 2.1.

1.3 Research Outline

When MCTS is applied to games with a combinatorial action space, the assumption that the action space should be enumerable ensures that performance is subpar. Alternative techniques which have been proposed that deal specifically with this problem, such as NMC and LSI, seem to outperform a MAB-based MCTS approach [31] and as of yet no formal extensions to MCTS exist which deal specifically with CMAB. However, these techniques have only been compared in the domain of μRTS [38] and no sensitivity comparison to the various features of the CMAB model has been performed as of yet. μRTS is a simplified implementation of a Real-Time Strategy (RTS) game with heterogeneous units. It is both deterministic and real-time, meaning that players can issue actions simultaneously, and actions are durative.

1.3.1 Problem Statement

From this the problem statement of this thesis follows as:

Determine the performance of state-of-the-art techniques in domains which can be modeled as a Combinatorial Multi-Armed Bandit (CMAB), and formulate extensions to MCTS which improve its performance in this domain.

1.3.2 Research Questions

From the perspective of MCTS, the following research questions arise from the problem statement:

1. *How can MCTS be extended to cope with a combinatorial action space?*
2. *How can MCTS be extended to exploit the side-information inherent in a CMAB model?*

To answer these questions an investigation into different representations of the action-space within MCTS is performed. Two variations on how the action-space can be explored within a CMAB are proposed, Hierarchical Expansion and Dimensional Expansion. From this problem statement also raises the question on which domains these techniques and extensions can be tested, and thus the following research questions:

3. *How does performance of the proposed techniques vary across variations in the CMAB model?*

To answer this question a domain is proposed in which we can easily experiment with the defining features of a CMAB. Experiments are then performed in an effort to analyze the sensitivity of the proposed algorithms to variations in the model and compared to state-of-the-art techniques LSI and NMC.

4. *What is the performance of the proposed techniques in the domain of Berlin?*

To address this final research question, the experimental setup is split up into four sections to evaluate the performance of the proposed extension in the domain of Berlin. The central line of comparison in these experiments is a top performing heuristic bot in the Berlin A.I. competition. The experimental setups analyze the topics of:

- **Action Space Representation**: The variations of the proposed extensions on MCTS are explored.
- **Algorithms**: A heads-up tournament in which the best performing MCTS extensions are compared to state-of-the-art techniques.
- **Partial Move Completion**: As the proposed action space representation allows partial moves to be returned by the search, a strategy needs to be defined to complete these moves. Here we compare the performance of both informed and un-informed strategies.
- **Playout Strategies**: How the performance of the algorithms is affected by the introduction of domain knowledge.

1.4 Thesis Outline

The thesis opens with an introductory chapter explaining the problem at hand and its background, supplemented with a cursory overview of the techniques explored within. **Chapter 2** provides further detail and analysis of the domains explored in this thesis and outlines the various approaches researched to deal with the domain of Berlin.**Chapter 3** details the techniques used in this thesis, the theory behind them and any alterations required for their application in the domains explored. **Chapter 4** outlines the enhancements proposed in this thesis and discusses their strengths and shortcomings. Finally, **Chapter 5** provides the results and analysis of the experiments in both domains on the outlined algorithms while **Chapter 6** concludes the thesis and offers suggestions of improvement and directions for future research.

2

Problem Domains

Chapter overview: *The following chapter details the problem domains in which experiments are run in this thesis. Two different problem domains are described; a straightforward representation of a CMAB, and the domain of Berlin.*

2.1 Combinatorial Multi Armed Bandit

The problem domain outlined in this section was devised to provide a test bed which was as close to the CMAB model as possible. By providing this domain the various techniques and enhancements outlined in this thesis could be tested without concern of abstraction variations, as detailed in Subsection 2.2.2.

The multi-armed bandit (MAB) problem, which serves as the basis of this model, is the problem of optimizing the sum of rewards earned through a sequence of actions. At each stage the agent has to execute one a_i from a set of stochastic actions $A = \{a_1, ..., a_k\}$. Each a_i is parameterized with an unknown distribution χ_i from which a reward is drawn if the agent executed a_i [38].

The primary distinction between the MAB and CMAB model is that for a MAB the agent needs to decide for a single variable, whereas in a CMAB it needs to decide for n variables. Formally, a CMAB is defined by:

- C: A set of values consisting of $\{A_1 \times ... \times A_n\}$, where A_i is the set $A_i = \{a_{i;1}, ..., a_{i;j}\}$. A single value from C is referred to as the combined action cA. A combined action consists of n variables, defined as the sub-actions a_i of cA.
- R: A reward distribution $\chi_1 \times ... \times \chi_n \to \mathbb{R}$, depending on sub-actions a_i within a given cA.
- V: A function which given a combination of values cA drawn from all A_i, returns whether the combination is legal and allowed within the domain.

A CMAB can be translated into a MAB by translating each legal value combination into a single arm. However, in doing so we lose the inherent structure of the values which can be exploited to learn the expected reward per sub-action a_i [38, 13].

To showcase the distinction of a linear versus non-linear correlation between the variables, a reward function is proposed in which the inter-correlation between variables can be adjusted. In this reward function the value returned by the reward distribution for a_i is multiplied by a given weight and then integrated through summation or integration with the value returned for a_{i-1}. Formally, this reward function is defined as:

$$v_0 = 1 \tag{2.1}$$

$$v_i = op_i(R(a_i) \times w_i, v_{i-1}) \tag{2.2}$$

where

w_i is the weight set per a_i

op_i is either $\prod$ or $\sum$ and set per a_i

During experimentation op_i is stochastically determined to be either $\prod$ or $\sum$ at the start of a game based on a variable ϵ representing the non-linearity of the reward function. To remove any introduced bias, the structure is determined at the beginning of each game, and shared between players. During experimentation R is represented by Gaussian distributions with $\mu = 0.0, \sigma = 2.0$. The game is played for a single round, with the highest scoring player being the player with cA which maximizes the reward function where R is replaced by μ, defined as the actual reward function per a_i.

2.1.1 Complexity Analysis

Action Space As all Bandits have an equal number of arms, the action space, or *branching factor* of this game is x^y, where x is the number of Arms, and y the number of Bandits in the game.

State Space As the game is played for a single round, the action space is equal to the state space.

2.2 Berlin

This section details the domain of Berlin [7] and provides a complexity analysis, followed by two alternative representations of the action space. These representations approach the action space from two different perspectives, one from a *node-as-agent* perspective, and the other as a *soldier-as-agent* perspective.

Berlin is a simultaneous N-player zero-sum game with similarities to the popular board game Risk, but fully deterministic. It is a turn based strategy game with homogeneous units in which the goal is to achieve the maximum number of points within a preset turn limit. The game is played on a directed graph which is fully observable, with soldiers only allowed to move along edges connected to their origin node. Each turn the player decides if and how soldiers are sent over adjacent edges to attack or reinforce neighboring nodes. Ownership of a node changes if there is a single attacker with more soldiers than the defender. The remainder in difference between the owner with the most soldiers and second-most soldiers on a node is the number of soldiers that survive for the next round. Each node specifies the number of victory points it is worth at the end of the game, and the number of reinforcements the owner receives per turn upon.

Within the experiments we model the simultaneous aspect in Berlin by viewing the deciding player as the first player, clearing any decisions made by other players. This in turn results in paranoid, but still adequate playing strength, as determined by [17, 50].

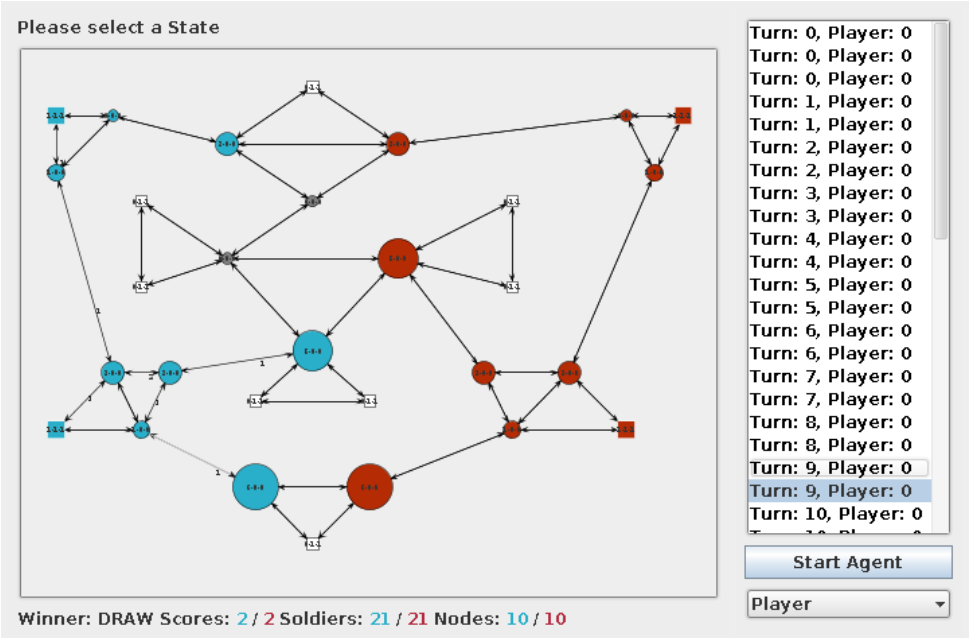

Figure 2.1: The GUI built to locally visualize and run Berlin tournaments.

2.2.1 Complexity Analysis

Action Space By viewing the action space complexity from the perspective of a node, the core complexity is a weak composition [39] constituting the number of soldiers that can be sent out on adjacent edges. Then, the variations of these compositions in which the number of soldiers that are sent out is varied needs to be explored. Both these facets combined define the summed compositional part of Equation (2.3). The final product in Equation (2.3) constitutes the combinatorial aspect of a decision for all nodes owned by the player.

$$\prod_{n \subseteq N} \sum_{k=0}^{n_s} \binom{n_e + k - 1}{k - 1} \tag{2.3}$$

where

N is the set of nodes owned by the player
n_s is the number of soldiers on node n
n_e is the number of edges connected to node n

While the action space complexity is not much of an issue in early game situations, the combinatorial nature ensures that the actions of any non-trivial sized state can no longer be efficiently enumerated. Per example, the graph in games in Berlin can easily consist of 30 nodes, with an edge connectivity of 3. Mid-game the average number of soldiers per node is approximately 10 for decent players. In a game where a player controls half the map, 7.006×10^{36} legal moves exist per turn.

Game Tree The complexity of the game tree follows naturally from the action space as Berlin is a game with a preset number of rounds, as outlined in Equation (2.4).

$$\prod_{i \subseteq r} A_i \tag{2.4}$$

where

A_i is the action-space complexity of round i
r is the number of rounds in a match, typically 25

It must be noted that a large number of actions do not lead to a distinct state, which in turn suggests that transposition tables could provide a benefit to any search based approach [2, 14, 46, 37].

2.2.2 Action Space Representation

Key to all algorithms which deal with representing the problem domain as a CMAB is how to define the division between CMAB and its child MABs. For Berlin we propose two alternative representations: *node-as-agent* and *soldier-as-agent*. The decision between both representations is a choice over the branching factor (width) versus depth of the search tree as shown in Figures 2.2, 2.4 and 2.3. Both NMC and LSI inherently benefit from a smaller branching factor. MCTS benefits from a balance between the two as no guarantees can be made on all sub-actions in a cA action being explored if the depth is too large, and the algorithm suffers if the tree is too wide. This problem is further explained in Subsection 4.4 and experimentally explored in Subsection 5.3.4

Node-as-Agent

In this representation we see each individual MAB as a decision for a single node. The perspective of the agent [30] is from the node, where per ply, all actions for a given node are expanded from the given state. The depth of the tree *per ply* then becomes $|N|$, the number of nodes, as it is factored out of the action space complexity Equation 2.3. The action space per layer, or branching factor of the tree, is considerably reduced but still sizeable as governed by the remainder of Equation 2.3.

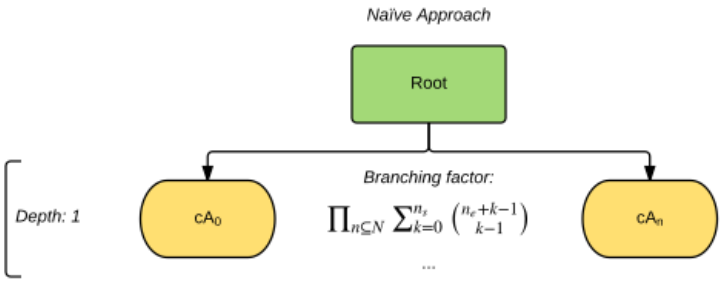

Figure 2.2: Naïve modeling of the action space in Berlin.

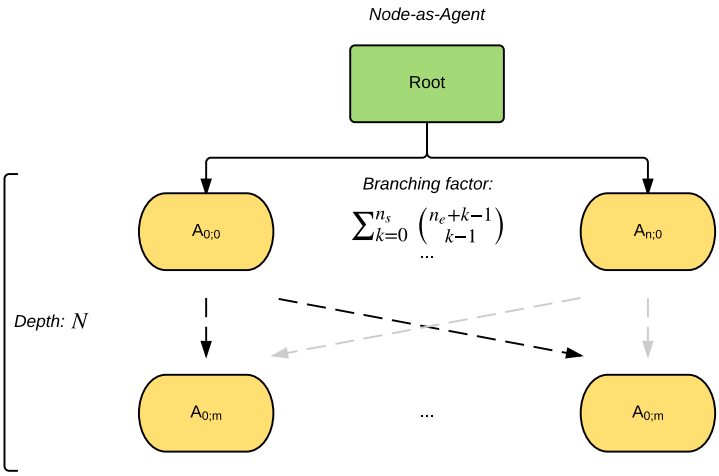

Figure 2.3: Node-as-Agent modeling of the action space in Berlin.

Soldier-as-Agent

In this representation we see each individual MAB as a decision for a single soldier. The depth of the tree *per ply* then becomes $|S|$, where S is the total number of soldiers for the target player. The action space per tree is then governed by the number of edges for the node the soldier resides on, which usually is $1 \leq n_e \leq 6$.

2.2.3 Heuristic Agent

To provide a performance baseline in the experiments in Berlin, the Genghis Khan agent from the online Berlin AI tournament has been recreated. This agent has performed well in the competition and was open-sourced so we can freely analyze its performance and integrate it into playout simulation. [7, 25]

Each turn, this agent assigns a value and constructs a distance matrix for each node. Each outgoing edge is then assigned a normalized value equal to the combined worth of the heuristic value of all nodes multiplied by their distance over this edge, given the distance matrix. Soldiers are divided among the outgoing edges according to their respective normalized value. Any remainder soldiers are divided equally among all edges. As a defensive strategy, the bot deducts the number of soldiers required to defend the node from the soldiers to be sent along the edges. The number required is defined as the number of enemy soldiers on directly adjacent nodes.

The value of a node is defined as:

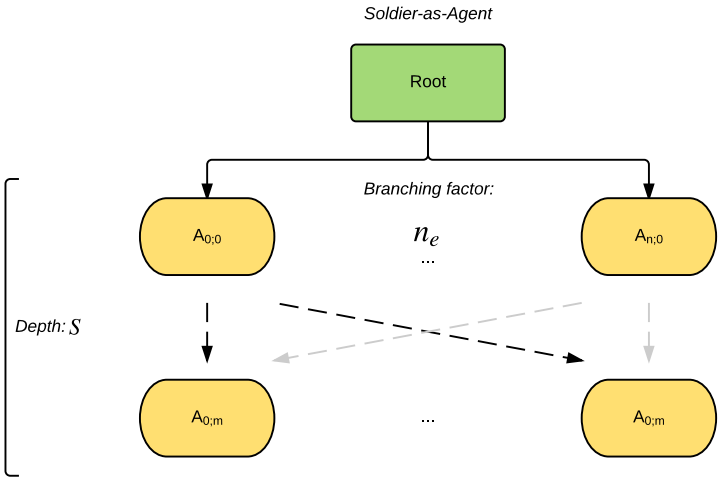

Figure 2.4: Soldier-as-Agent modeling of the action space in Berlin.

$$owner(n) \times 2n_{vp} \times 4n_{sp} \tag{2.5}$$

where

n is node in question

$owner()$ is a weighted function returning -1 for owned nodes, and 1 for enemy and unowned nodes.

n_{vp} is the number of victory points the node is worth at the end of the game.

n_{sp} is the number of soldiers the node produces per round upon ownership.

3

Search Algorithms

Chapter overview: *This chapter outlines the various exploration strategies, search techniques and extensions used in this thesis. We first elaborate on the various exploration strategies that are employed by the search techniques explored in this thesis. We then detail the search techniques evaluated in this thesis, which comprise Monte-Carlo Tree Search (MCTS), Naïve Monte Carlo (NMC) and Linear Side Information (LSI).*

3.1 Exploration Strategies

One of the core concepts in search algorithms is the policy which decides which node is explored at each ply. These policies determine the nature of the search algorithm in terms of how it responds to cumulative and simple regret. *Simple regret* is the expected error between an algorithm's recommendation, and optimal decision [9]. The *cumulative regret* is the accumulated error between an algorithm's recommendation and the optimal decision over multiple actions [9]. Within the search strategies explored in this thesis UCB and Sequential Halving are used, and thus explained in this section. We elaborate on Successive Rejects, as it served as an introduction on the concept of discarding arms from future consideration.

3.1.1 Upper Confidence Bounds

Upper Confidence Bounds (UCB) is a finite time exploration strategy for MABs proposed by [5]. The basis of the algorithm is that it optimizes cumulative regret at an optimal logarithmic rate, without knowledge of the reward distributions R behind each arm. The policy is split up into the current average reward μ_c, and the one-sided confidence interval for μ_c. UCB gives an upper bounding value for each arm, below which the expected reward drops with high probability [5]. For n samples, UCB has an optimal rate of cumulative regret of $O(ln(n))$ [33]. This in turn means that any action a_i recommended suffers a simple regret that decreases at best at a polynomial rate [9]. Algorithm 1 outlines the UCB selection policy.

3.1.2 Successive Rejects

As the name implies, Successive Rejects successively removes the worst performing arm from the selection. Each round a budget n_k is computed, each remaining arm is sampled, and finally the worst arms removed. The lower bound on simple regret is guaranteed by by the lengths of the rounds [4]. Algorithm 2 outlines the Successive Rejects policy.

3.1.3 Sequential Halving

As with Successive Rejects, the algorithm is divided into rounds in which arms are rejected from further rounds. The prime distinction from Successive Rejects is how time is divided among the rounds, and which arms are rejected per round. In Sequential Halving each round the number of arms are halved, and the sample budget is distributed equally among all rounds, and then among all arms [27].

Algorithm 1 Upper Confidence Bounds (UCB) [5]

Input: Sample Budget T, Set of Arms A
Output: Recommended arm a_i

// Ensure that each arm has at least 1 sample

for all $a_i \leftarrow arms$ **do**
    $\bar{x_i} \leftarrow sample(a_i)$
    $n_i \leftarrow n_i + 1$

// Exploit: Sample the best scoring arm, and update

for $t = 1 \dots T$ **do**
    $a_i \leftarrow max(\bar{x_i} + \sqrt{\frac{2ln(t)}{n_i}})$
    $\bar{x_i} \leftarrow \bar{x_i} + sample(a_i)$
    $n_i \leftarrow n_i + 1$
return a_i *where* $max(\bar{x_i})$

Algorithm 2 Successive Rejects [4]

Input: Sample Budget T, Set of Arms A
Output: Recommended arm a_i

$S_1 \leftarrow A$, $n_0 \leftarrow 0$
$\bar{logK} \leftarrow \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$

// Calculate a budget per round

for all $k \in \{1, \dots, |A| - 1\}$ **do**
    $n_k = \lceil \frac{1}{\bar{logK}} \times \frac{T - |A|}{|A| + 1 - k} \rceil$

// Each round, reject the weakest arm

for $k = 1 \dots |A| - 1$ **do**
    for all $a_i \in S_k$ **do**
        $v_i \leftarrow v_i + sample(a_i, n_k - n_{k-1})$
    $\bar{\mu_i} \leftarrow \frac{v_i}{|S_k|}$
    $S_{k+1} \leftarrow max(S_k, |S_k| - 1)$
return S_k, *containing a single element.*

Sequential Halving outperforms Successive Rejects and currently provides the best bounds on simple regret [27]. It was adapted for MCTS in a variant named Sequential Halving On Trees (SHOT) [11], its application as a selection policy for tree search is explored in [40]. Algorithm 3 outlines the Successive Rejects policy.

Algorithm 3 Sequential Halving [27]

Input: Sample Budget T, Set of Arms A
Output: Recommended arm a_i

$S_1 \leftarrow A$, $rounds \leftarrow \lceil log_2 K \rceil - 1$
for $k = 1 \dots rounds$ **do**
  $samples \leftarrow \lfloor \frac{T}{|S_k| \lceil log_2 |S| \rceil} \rfloor$
  for all $a_i \in S_k$ **do**
    for $s = 1 \dots samples$ **do**
      $\mu_i \leftarrow sample(a_i)$
  $S_{k+1} \leftarrow max(S_k, \lceil \frac{S_k}{2} \rceil)$
return S_k, *containing a single element.*

3.2 Monte-Carlo Tree Search (MCTS)

Monte-Carlo Tree Search (MCTS) [16] is a best-first search technique in which a tree is iteratively built consisting of nodes representing states. Heuristic evaluation of a game state is not performed as traditionally, using the current state of the world, but on the simulated terminal state resulting from the selected state. This makes MCTS uniquely qualified for problems where determining an optimal course of action using heuristics is non-trivial [12]. MCTS consists of four phases [12], as depicted in Figure 3.1; each repeated in succession until a computational constraint is reached, commonly being wall-clock time. These phases are:

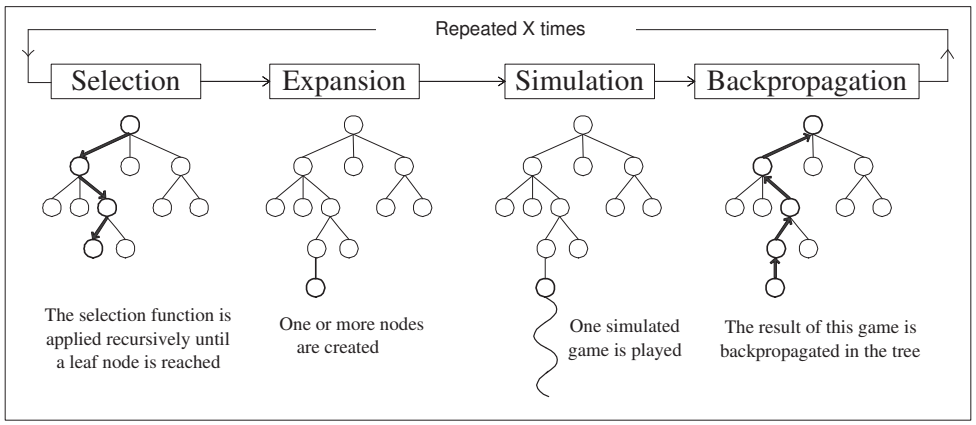

Figure 3.1: Monte-Carlo Tree Search

Selection: In the selection phase, starting from the root node, a child node is recursively selected according to a selection strategy S_s until a leaf node s is reached.

Expansion: In the expansion phase, the expansion strategy S_e is applied to the selected node s, which traditionally expands the node by populating it with subsequent states.

Simulation: Starting from leaf node s, the game is simulated in self-play according to some playout strategy S_p until the cut-off strategy S_c decides to stop the simulation, traditionally when a player has won the game. Such a simulation is called a *playout*.

Backpropagation: In this phase the results from the simulation is propagated backwards from leaf node p to the nodes that were traversed to reach p according to a back-propagation strategy S_b. Traditionally the result is counted singular positive if the root player has won, and singular negative for a loss [12, 44].

MCTS has successfully been applied to similar domains in which tactical planning is central [6, 47].

3.2.1 RAVE

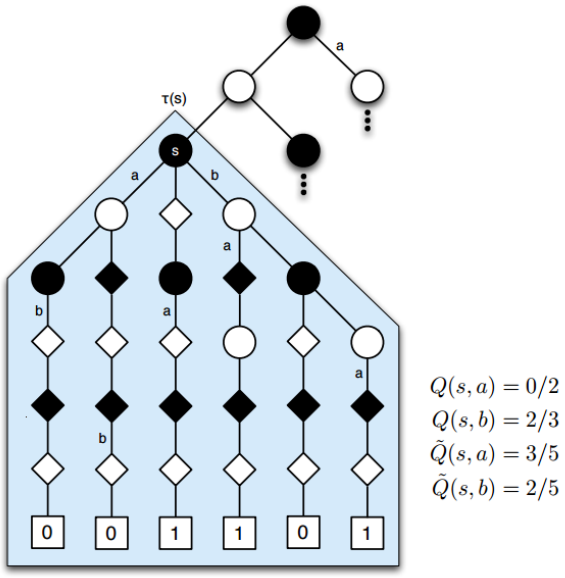

Figure 3.2: Example of action a & b estimation from state s. The outcome of the simulation is shown in the bottom squares. Monte-Carlo estimation favours move b, as a led to two losses. However, move a also led to three wins, and is favoured by RAVE. The simulation with move a from the root does not belong to subtree (s) and does not contribute to the AMAF estimate $Q(s, a)$ [24].

Monte-Carlo Tree Search does not generalize related positions or moves. The simulations over a state, or action, are the only way in which the algorithm determines their value. The Rapid Action Value Estimation (RAVE) [23] algorithm provides an initial value for a given state or action which decreases over time as simulation information on it becomes available. This value is gained through knowledge shared between related nodes in the search tree. It significantly improves performance in a domain where states or actions are recurring and can be shared between subtrees.

The base assumption of RAVE is that the value of an action a in state s is equal for all states within the subtree $\tau(s)$, regardless of when it is played. This assumption, that the value of a move is unaffected by other moves, is often called the *All-Moves-As-First* heuristic. RAVE works by storing the mean outcome of all simulations in which action a is selected after state s, for state s [23]. In traditional application of RAVE in Upper Confidence Bounds (UCB) [16] applied to Trees (UCT) [16], the value returned by UCT and RAVE are combined linearly with a decaying value β based on the number of simulations already run through the node. This ensures that the UCT value of the node is taken in the limit, instead of the biased estimate returned by RAVE.

3.3 Naïve Monte-Carlo Sampling (NMC)

Both Naïve Monte-Carlo (NMC) [38] and Linear Side-Information (LSI) [31] are based on the theory that to deal with an expansive action space as proposed in online planning in a CMAB, an algorithm needs to define two strategies:

1. A *candidate generation strategy*, for reducing the set of all possible combined actions cA, named C, leading from a given state to a reasonably small subset $C^* \subseteq C$ of candidates.
2. A *candidate evaluation strategy*, for identifying the best combined action cA in C^* by gradually improving the corresponding estimates in μ.

As outlined in the section on the CMAB model, a combined action cA is defined as an action consisting of several sub-actions. Each set of possible sub-actions is defined as A_i, with a single sub-action defined as a_i. The combination of these sub-actions is what leads to the expansive size of C [31].

Naïve Sampling is a sampling strategy for a CMAB based on the assumption that the reward distribution can be decomposed as a linear summation: $R(x_1, \ldots, x_n) = \sum_{i=1\ldots n} R_i(x_i)$. This assumption, that the reward distribution is linear over the domain, is called the *naïve assumption* [38]. Within NMC, this assumption allows the CMAB problem to be broken into a collection of $n + 1$ MAB problems:

- MAB_g, the *global MAB* where each of the legal combined actions cA that have been sampled so far is represented as an arm.
- MAB_i, the *local MABs*, one for each set of sub-actions A_i which only consider the sub-actions $a_{i;j} \in A_i$.

Naïve Sampling uses the local MABs to *explore* new value combinations that have a higher expected reward through the naïve assumption. The global MAB is used to exploit the value combinations with the best reward so far [38]. Figure 3.3 shows a schematic version of the NMC algorithm. At each round t a policy π_0 is used to determine whether to *explore* (MAB_i) or *exploit* (MAB_g) [38].

- If *explore* was selected: $A_1, \ldots, A_n$ is sampled by using a policy π_l to select a value for each a_i independently. The resulting combined action cA is added to the global MAB_g.
- If *exploit* was selected: $A_1, \ldots, A_n$ is sampled by using a policy π_g to select a value combination using MAB_g.

For π_0 traditionally an ϵ-greedy strategy is chosen, where *explore* is chosen with probability ϵ, and *exploit* with $1 - \epsilon$. Assuming $\epsilon < 1$ every $cA \in C$ will be generated and then sampled infinitely often. Thus, NMC converges to the best $cA \in C$, independently of whether the naïve assumption holds [31].

NMC was shown to outperform the standard tree search algorithms, such as UCT and ABCD in the domain of μRTS when the action space was not restructured.

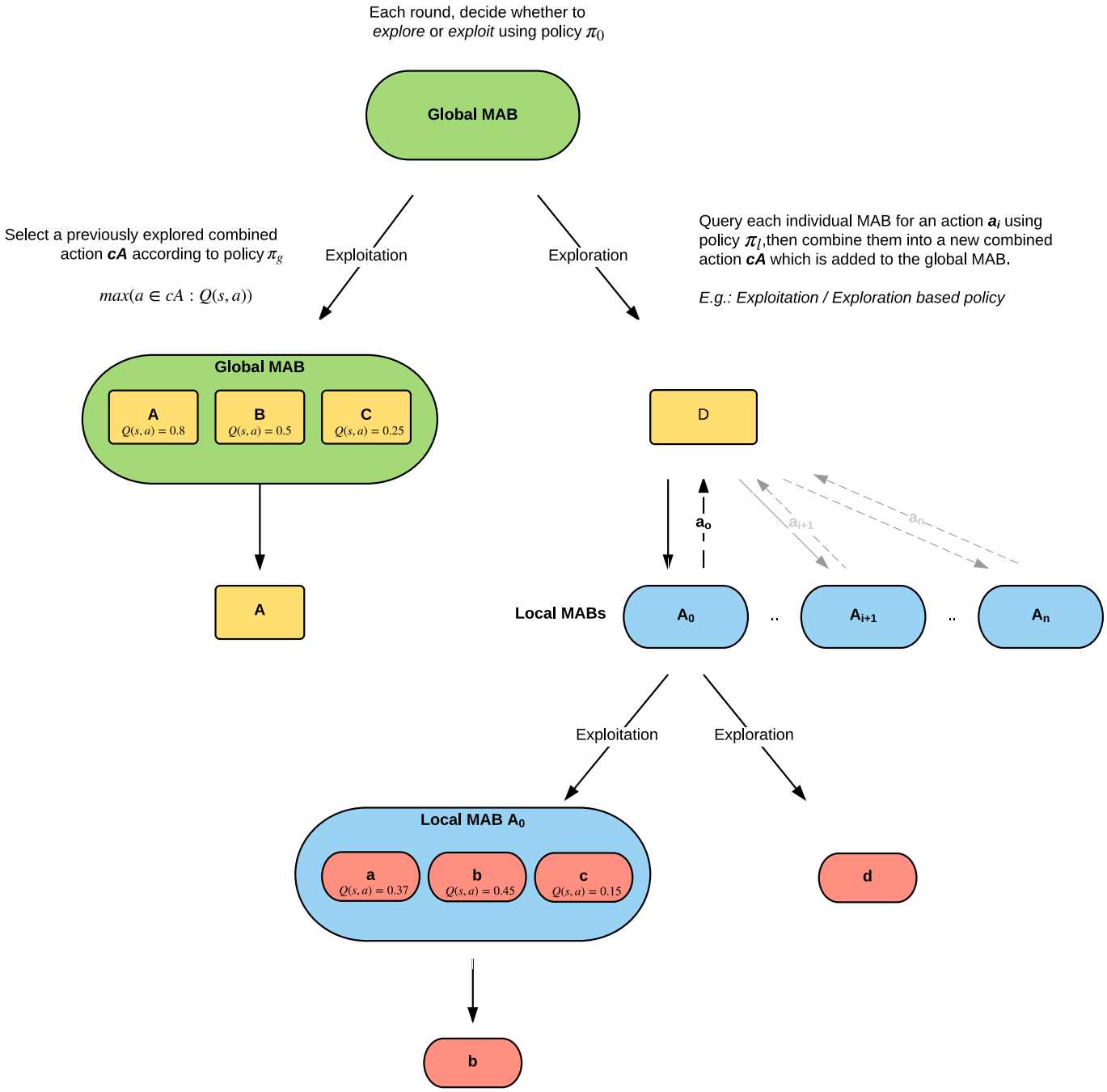

Figure 3.3: Naïve Monte Carlo,
where π_l uses an *exploitation/exploration* policy to select a sub-action a_i for local MAB A_i.

3.4 Linear Side-Information (LSI)

The success of NMC was the starting point of LSI [31]. In NMC the candidate generation and evaluation are stochastically interleaved, at a single sample resolution. This interleaving allows the *exploiting* of the samples made at evaluation in an effort to improve the estimate side information for *exploration*.

However, if m samples are devoted directly to candidate generation, m new candidates are generated. If the naïve assumption holds, these could instead be generated with guidance from the side information [31]. Also, while NMC converges to the best cA, for no reasonable budget of samples T can a guarantee be made on the quality of the recommended action through the set of C, or the generated subset of C^* [31]. This latter observation stems from the fact that the recommended action by NMC is selected through the empirical mean which could be based on a single sample of the respective combined action cA [31].

LSI tries to address both of these issues. The algorithm handles candidate generation and evaluation sequentially, instead of interleaved. The overall sample budget T is divided into T_g, for the candidate generation *which runs first*, and $T_e = T - T_g$, for the candidate evaluation *which runs second*. This alteration is based on the fact that for regular planning in MABs state-of-the-art algorithms guarantee an *exponential* decrease of probability in selecting a sub-optimal cA with a given T [31].

Candidate generation works by first constructing a weight function over all sub-action dimensions A_i. This is done by evaluating the sub-actions within each dimension using sampling budget T_g. Using these weight functions a distribution is constructed for each dimension A_i. k combined actions are then constructed by sampling the distribution behind each A_i and combining the resulting sub-actions into a combined action. Candidate evaluation then uses the Sequential Halving policy given the sampling budget T_e to determine the best combined action cA.

Figure 3.4 depicts a schematic overview of the two-phase sampling scheme which serves as the basis for LSI, Figure 3.5 shows the pseudo-code of the entire algorithm. More formally, the algorithm is divided up into the following phases:

- Generate which serves to generate the first $k(T_e)$ cA-actions. This procedure is split up into:
  1. The generation of a weight function $\hat{R}$ over the sub-actions in A_i. R is constructed by creating a combined action cA for each individual sub-action $a_{i;j}$. The combined action is constructed using a given policy E and evaluated using part of the sampling budget T_g.
  2. The generation of a biased distribution $D_{\hat{R}}$ over C towards $\hat{R}$.
  3. Sampling (up to) $k(T_e)$ cA-actions from C using $D_{\hat{R}}$.
- Evaluate which uses the Sequential Halving algorithm detailed in Subsection 3.1.3 until a single combined action cA is left to be recommended.

The policy E used to construct a combined action from a targeted sub-action is what provides the two main variations of LSI: Linear Side Information for Vertices (LSI_v) and Linear Side Information for Facets (LSI_f). The distinction in the two variations lies in how the sub-actions used for sampling are extended to full actions:

- In LSI_v, all m samples in the budget for $a_{i,j}$ are dedicated to the cA-action containing only $a_{i,j}$. The other sub-actions are chosen to be *null*-moves.
- In LSI_f, the budget m is randomly distributed over all cA-actions which contain $a_{i,j}$. In other words, m cA-actions are generated in which all sub-actions beside $a_{i,j}$ are randomly selected.

Depending on the linearity of the reward function in the domain, and the possibility to use *null*-moves, either LSI_v or LSI_f is preferred. If the reward function in the domain is linear in nature, LSI_v is preferred as the underlying value can be calculated without regard of the other sub-actions. If it is non-linear in nature, LSI_f is preferred as the average over all other sub-actions needs to be considered to provide a correct estimate of the average value of the sub-action at hand.

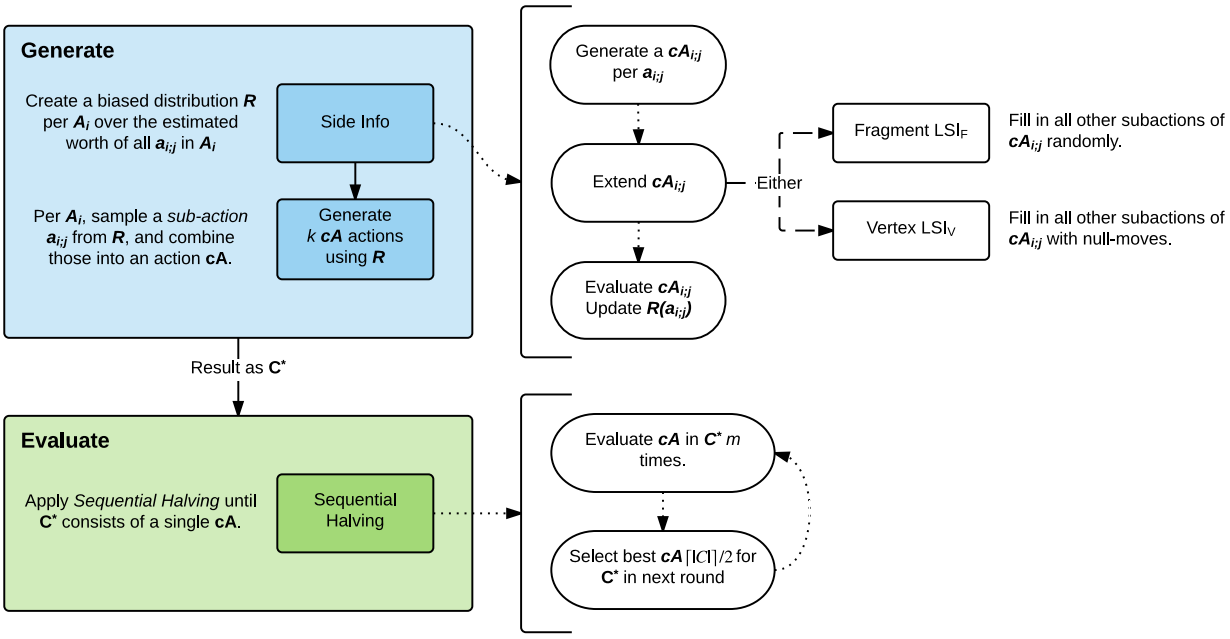

Figure 3.4: Linear Side Information (LSI)

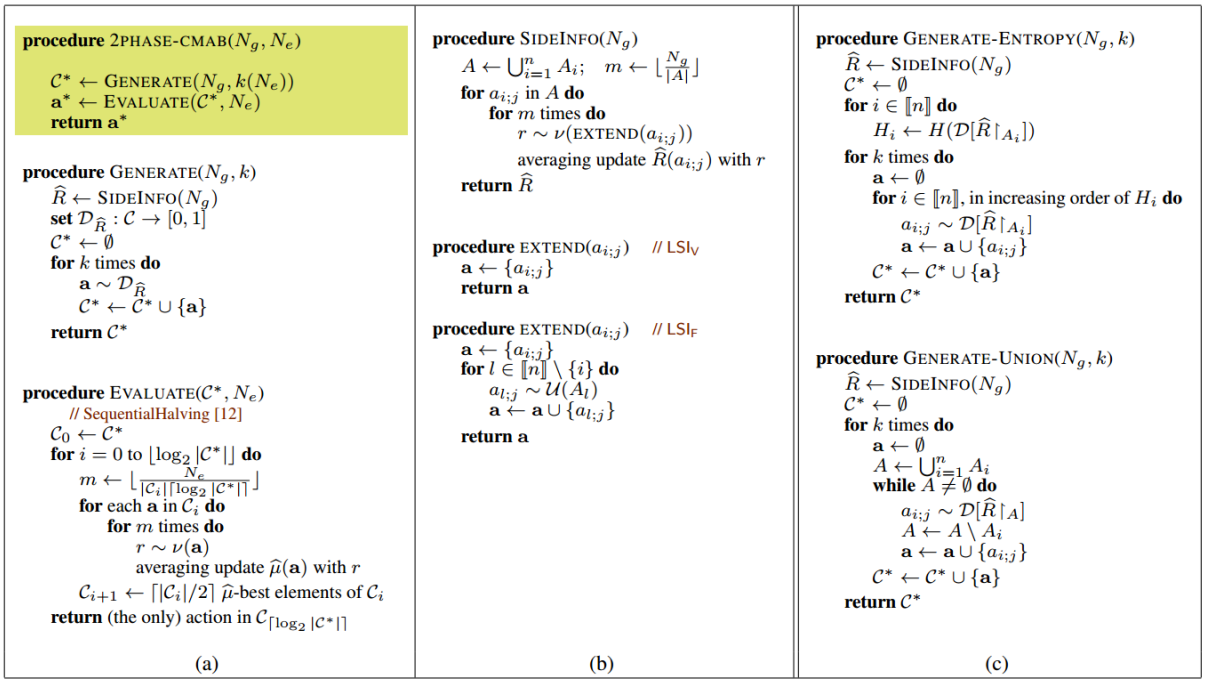

Figure 3.5: (a) The general 2PHASE-CMAB planning scheme, as well as the LSI scheme for candidate generation and evaluation, (b) specifics of the LSIV and LSIF instances of 2PHASE-CMAB, and (c) two specific procedures for LSI candidate generation. [31]

3.4.1 Time Limited LSI

The algorithm outlined in Figure 3.5 is limited only in the sample budget T, and cannot be cut-off based on a time-constrained budget. As Berlin, one of our problem domains, is setup as a time-constrained problem, an alternative is suggested in which T_e and T_g represent a budget in calculation time. For this, the EVALUATE and SIDEINFO functions were altered as shown in algorithms 4 and 5.

Algorithm 4 LSI^{tl} Evaluate

Input: Set of generated c actions C^*, time budget T_e
Output: Recommended c action

$C_0 \leftarrow C^*$
for $i = 0 \dots \lfloor log_2|C^*| \rfloor \wedge timeLeft(T_e)$ **do**
  $m \leftarrow \lfloor \frac{T_e}{|C_i|\lceil log_2|C^*| \rceil} \rfloor$
  for all $c \in C_i$ **do**
    for m times $\wedge$ $timeLeft(T_e)$ **do**
      $r \sim v(c)$
      averaging update $\hat{\mu}$ with r
  $C_{i+1} \leftarrow \lceil |C_i|/2 \rceil$ $\hat{\mu}$-best elements of C_i
  $sort(C_{i+1})$ using $\hat{\mu}$.
return $max(C_i)$ using $\hat{\mu}$

Algorithm 5 LSI^{tl} SideInfo

Input: Time Budget T_g
Output: Recommended arm $\hat{R}$

$A \leftarrow \cup_{i=1}^{n} A_i$
$m \leftarrow \lfloor \frac{T_g}{|A|} \rfloor$
$i \leftarrow 0$
while A_i contains unsampled action $a_{i,j} \wedge timeLeft(T_e)$ **do**
  for m times $\wedge$ $timeLeft(T_e)$ **do**
    $r \sim v(\text{EXTEND}(a_{i,j}))$
    averaging update $\hat{R}(a_{i,j})$ with r
  $i \leftarrow modulo((i+1), |A|)$
return $\hat{R}$

Experimental results showed both LSI and LSI^{tl} to be of equal playing strength.

4

Proposed Enhancements

Chapter overview: *The main contributions of this thesis: an alternative representation of the search space which allows a combinatorial action space to be explored using MCTS, named Hierarchical Expansion (HE), and an extension on this technique which allows MCTS to determine the optimal sequence of dimensions to explore, named Dimensional Expansion (DE).*

4.1 Introduction

The core issue for search techniques in complex action spaces is the inability to efficiently enumerate the actions possible [38], and is the issue at hand which the proposed alterations try to tackle. However, inspiration for the extensions outlined in this chapter came from the intuitive understanding that a CMAB consists of MABs with heterogeneous payoff. Move ordering such as Killer Heuristics have shown to produce increased playing strength in the context of $\alpha\beta$ search [1]. From this understanding follows that the sequence in which MABs are explored could provide a move ordering of sorts, and that there exists some optimal sequence which leads to an optimal increase in playing strength.

Furthermore, MCTS is a best-first search algorithm that thrives in problem domains which have ample "pruning opportunities". In other words, the easier it is for the algorithm to distinguish between "good" moves, the deeper it will search the tree. If the value of a move can be shown to be better than other moves at the same ply, MCTS with UCT will tend to focus on exploration of those moves. Conversely, if competing moves exist with similar value, the algorithm tends to divide the computation budget between them. Restructuring a search space to provide such "pruning opportunities" through Move Groups has led to an increase in playing strength in both Amazons and Go [45, 14, 36].

Both Hierarchical Expansion (HE) and Dimensional Expansion (DE) follow a similar strategy to Move Groups to deal with an expansive action space, namely that the depth of the tree is expanded in an effort to curb the branching factor [14, 53]. The primary distinction between HE and DE is the application of domain knowledge. With HE we assume to know some heuristic function that orders the dimensions to be explored. With DE we assume no knowledge on the domain, and the search space is structured such that MCTS will simultaneously search for both the optimal sequence of dimensions to explore and move to return. From this follows the intuitive understanding that if the heuristic function of HE is optimal, HE will outperform DE.

As such, HE and DE provide two benefits:

1. They allow MCTS to operate on a prohibitively large action space by re-organizing the action space such that the branching factor is curbed.

2. They introduce a pruning opportunity in the tree by placing those nodes first which are most likely to have a distinct value.

Subsection 4.2 details the Hierarchical Expansion algorithm, Section 4.3 details the Dimensional Expansion algorithm. Figures 4.1, 4.2 and 4.3 showcase the different expansion functions on a sample domain with a 3-dimensional (Yellow, Green & Blue) Combined Action, in which each dimension has 2 actions (0 & 1).

4.2 Hierarchical Expansion

The basis of both the HE and DE is to sacrifice the depth of the tree to curb the branching factor of the tree. In HE and DE we assume that the action space can be divided into a set *dimensions* D, each $a_i \in D$ with a set of actions A_i. For HE we then assume some ordering for D is known, or that the order in which we explore dimensions does not matter.

In normal expansion for MCTS a combined action cA is created which consists of a choice for all dimensions in D. In Hierarchical Expansion we create a sub-action a_i and select an action for a single dimension $d_i \in D$. At each iteration we expand into a sub-action and merge this into the incomplete combined action cA constructed thus far until a choice is made for all dimensions of D.

Figure 6 shows the algorithm for Hierarchical Expansion within MCTS, while Figure 4.2 shows a schematic representation of the example domain expanded using HE. By restructuring the action space we limit the branching factor to $\max |A_i|$, avoiding the construction of a combinatorial action space.

If the domain we are dealing is a recursive CMAB (A CMAB consisting of CMABs instead of MABs), we apply HE recursively on the sub action a_i. The incomplete combined action cA will then not be expanded further until we have made a complete decision cA_1 for all dimensions in the child CMAB.

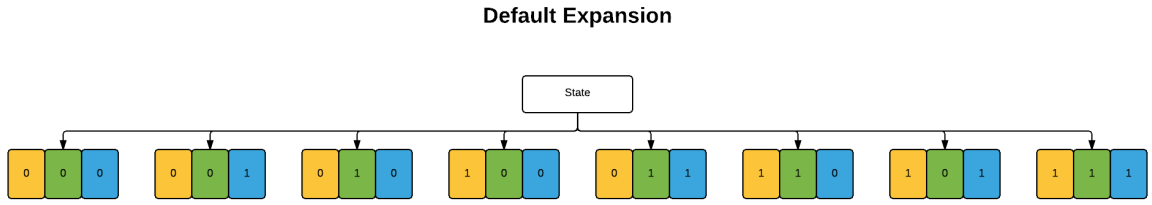

Figure 4.1: Naïve expansion of a combinatorial action for the given sample domain.

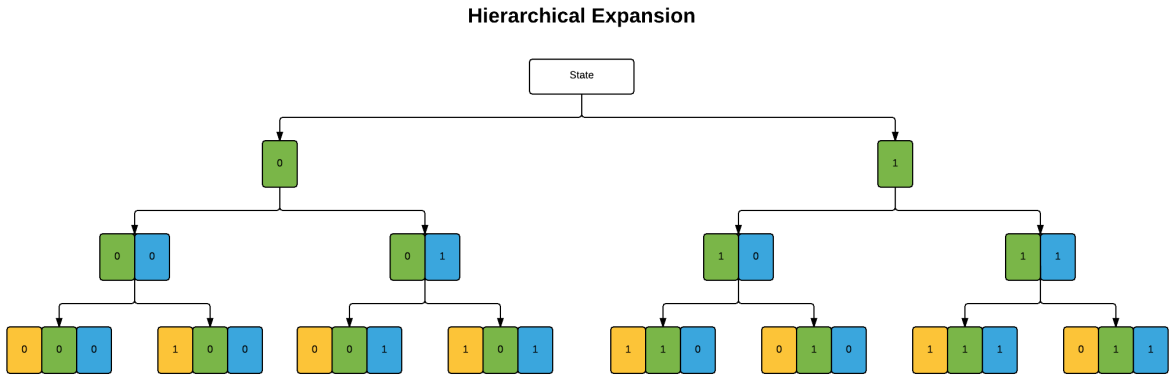

Figure 4.2: Hierarchical expansion of the action space

Algorithm 6 Hierarchical Expansion

Input: State s, Incomplete Combined Action cA, Set of Dimensions D, Ordering over Dimensions o.
Output: Set of (in)-complete Combined Actions possible for dimension x

1: $fD \leftarrow D - filledDimensions(cA)$ // The ordered set of dimensions with no action in cA
2: x $\leftarrow o(fD)$.head
3: **for all** $a \leftarrow Actions(s, x)$ **do** // The actions in dimension x given state s
4: $actions \leftarrow (cA \cup a)$
5: **return** *actions*

4.3 Dimensional Expansion

Key to HE is the ordering D on the dimensions. If this ordering is optimal, HE will be the optimal strategy. However, when this ordering is unknown, a situation very similar to $\alpha\beta$-pruning could occur in which little to no pruning occurs due to the fact that sub-trees have similar estimated value and no definitive decision can be made without extensive exploration.

To solve the problem where no ordering is known, Dimensional Expansion (DE) is introduced. This expansion strategy simultaneously explores over the individual dimensions and ordered sequence of dimensions. By sacrificing tree depth we hope to discover those nodes which provide early pruning opportunity. The algorithm works by sequentially choosing which dimension d_i to explore, and then to determine which action in d_i to explore. This decision is divided over two ply so that the algorithm may prune those aspects which are not interesting, and simultaneously avoid the combinatorial complexity of presenting both choices in a single ply.

The strength of the algorithm becomes apparent in those domains in which there is strong synergy in the value outcome of actions over different dimensions, in other words those domains with a non-linear component to their reward function. Through its exploration over the sequence of dimensions it will discover those sequences of actions which result in a higher expected reward.

The sequence in the action space is a concept introduced to allow exploitation of more valuable sub-actions, but in terms of the reward function this ordering over sub-actions does not exist. Thus, the action space is essentially duplicated over those branches which explore different sequences of dimensions. Two sequences with the same sub-actions give the same result, and the downside to this technique becomes apparent. Many samples are spent on branches which, after exploitation, are no longer visited and the samples spent on sub-actions in these branches are wasted. The recovery of this side-information is an active topic, and explored in experiment 5.3.4 where it is recovered through partial move completion with the application of RAVE [46, 24].

Figure 4.3 shows the expansion tree in the sample domain. In this example, the sub-trees which do not directly branch form the main line are implied as to limit the figure to a manageable level. The pseudo-code is outlined in Algorithm 7.

Algorithm 7 Dimensional Expansion

Input: State s, Incomplete Combined Action cA, Set of Dimensions D
Output: Set of (in)-complete Combined Actions leading from cA

```
1: if cA has no target dimension then
2:     fD ← D − filledDimensions(cA)        // The ordered set of dimensions with no action in cA
3:     a ← cA
4:     for all d ← fD do
5:         a.target ← d
6:         actions ← a
7: else
8:     x ← cA.target
9:     for all a ← Actions(s, x) do          // The actions in dimension x given state s
10:        actions ← (cA ∪ a)
11: return actions
```

4.4 Partial Move Completion

By allowing expansion over partial moves, or sub-actions, MCTS can no longer guarantee that a fully developed action will be returned. We therefore need to provide an Extension policy e which will, given a set of sub-actions, construct a valid combined action.

The options explored in this thesis outline the following policies:

- e_r: A stochastic policy which extends the incomplete move by filling it with a random selection for each $A_i \in undecided(cA)$.

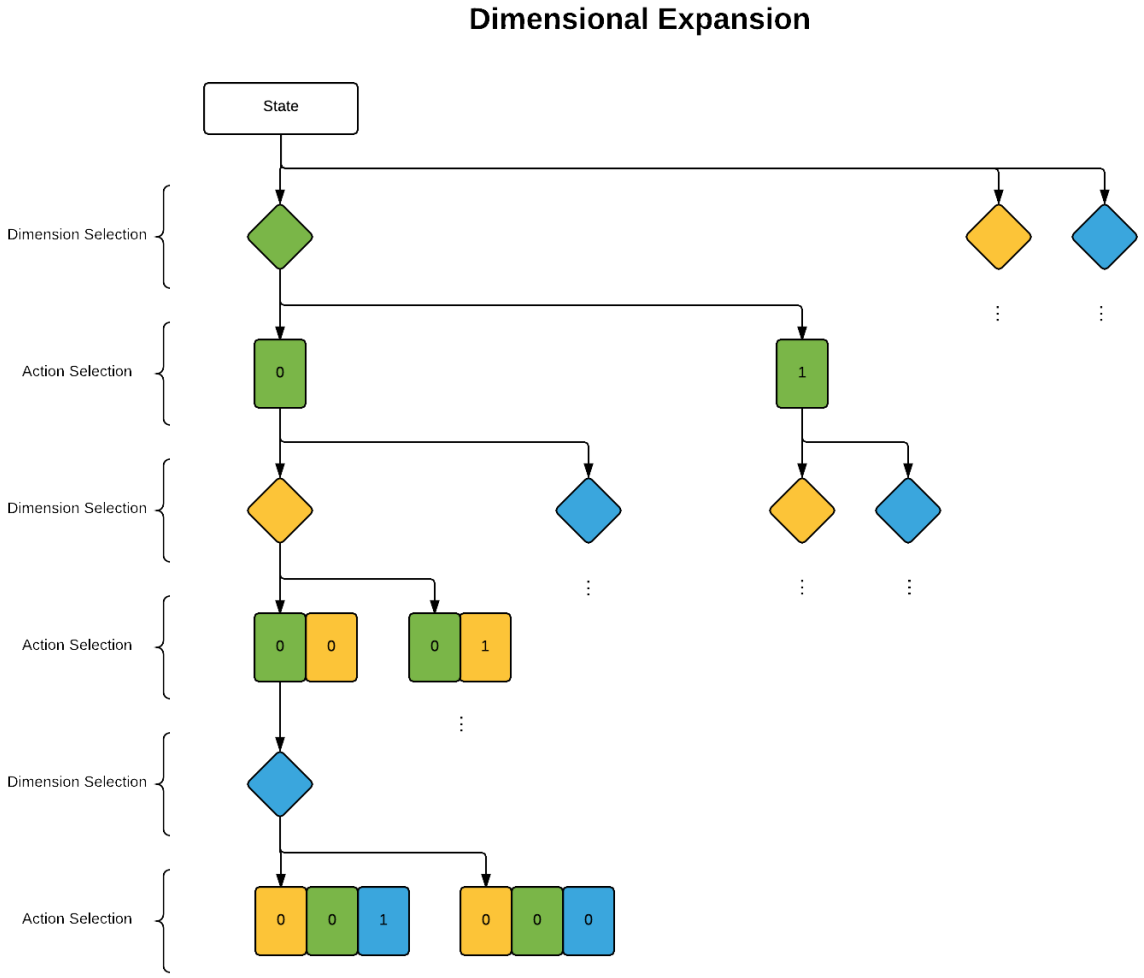

Figure 4.3: Dimensional Expansion. The sub-trees which do not directly branch from main line are not shown.

- e_k: A heuristic policy which extends the incomplete move by filling it with recommendations from a heuristic agent for each $A_i \in undecided(cA)$.
- e_{RAVE}: An online training policy which relies on RAVE to provide the best sampled sub-action during tree exploration for each $A_i \in undecided(cA)$. The RAVE estimated values from the root node are used to provide the values for the sub-actions.

Through the use of RAVE, or for this context a global action transposition table, we use the concept of side-information in the context of partial move completion. Both this and the option of introducing domain knowledge for impartial moves could considerably improve performance and is a concept experimentally tested in Section 5.3.4.

5

Experiments & Results

Chapter overview: *The following chapter contains an outline of the experimental setup, and the various experiments explored. For each experiment, the results are summarized and discussed. Detailed coverage of the results can be found in the relevant appendices.*

5.1 Experimental Setup

To conform to the experimental setup used by [31], an ϵ-greedy policy was used for all policies in NMC with the ϵ_0 parameter preset to 0.25. The T_g and T_e parameters for LSI were set to $0.25N$ and $0.75N$ respectively. The C constant, used by UCT (Algorithm 1) is set to the default $\frac{1}{\sqrt{2}}$. Early experimentation lead to decent values around 0.7 0.8, which lead to the setting of $\frac{1}{\sqrt{2}}$.

5.2 Combinatorial Multi-Armed Bandit

The first experiment is setup to showcase the sensitivity of various algorithms to action space complexity. In this experiment, we compare the following algorithms on the CMAB problem domain:

- Naïve Monte-Carlo (NMC)
- Linear Side Information Fragment (LSI_f)
- Monte-Carlo Tree Search using UCT (MCTS)
- Monte-Carlo Tree Search using UCT, Hierarchical Expansion (MCTS - HE) with an arbitrary ordering.
- Monte-Carlo Tree Search using UCT, Dimensional Expansion (MCTS - DE)

Both the weights w_i and range of the arms χ_i is set to 500. The ratio of non-linear versus linear operations op_i, ϵ, in the evaluation function is set to 0.1. The reasoning behind this is that we want to simulate real-world games, and any decently complex enough combinatorial game will have components in its value function which affect others (non-linearity). As the structure is reset at the beginning of every game, by using a small ϵ value we allow both games to be created which are purely linear in nature, and those which have non-linear aspects. As the value returned by each arm is also randomly determined at the beginning of the game, we weight the arms with a preset value of 500 to ensure that the resulting range of values returned by the combined actions C on CMAB is distinctive enough in the game tree.

As the number of bandits increases, the average value of the value function increases. From this follows that an algorithm with consistent playing strength should show an increase in value as complexity of any of the variables in the action space increases. For Hierarchical Expansion an arbitrary ordering ($0\dot{N}$) is picked. The bias introduced by this ordering is removed as the underlying structure of the CMAB is changed for each game.

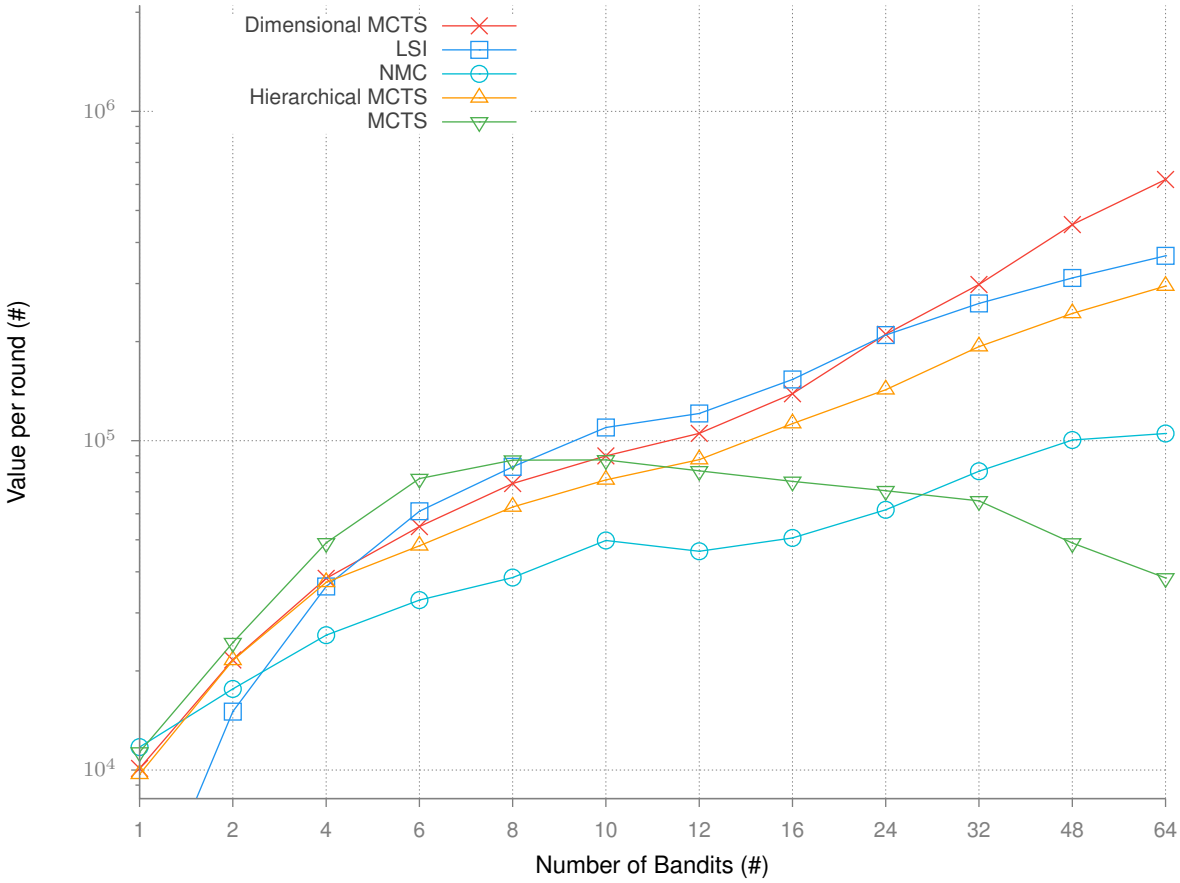

Figure 5.1: Results for the 4-armed bandit setup

Each datapoint in the experiment is the average result of 300 games, with a 5000ms timelimit per action. For LSI, we used the original algorithm which does not support time constraints. As such, the playout budget given is calculated per setup using simulated annealing to conform to the time limit.

5.2.1 Sensitivity, number of Bandits

Figure 5.1 shows the sensitivity of the various algorithms on a 4-armed bandit setup over the number of *Bandits*. The 4-Armed Bandit setup was chosen as the initial setup to showcase the performance of the various algorithms in a domain where the branching factor starts at a trivial size and gradually progresses to a sizable amount.

MCTS is the best performing algorithm until the action space of the domain reaches a threshold complexity of 8 bandits (i.e.: with an action space complexity of 4^8, or $65,536$). Performance then decreases drastically, given the fact that we expect a rise in value, not a decline. Here we see that the algorithm is ill-suited to deal with the rise in complexity and nears performance levels associated with random play as the action space cannot be enumerated any more.

While LSI underperforms as the complexity of the action space is low, it outperforms NMC once complexity starts to rise, which is in line with results shown in [31]. Both algorithms outperform MCTS as complexity rises, and perform as expected given the increase in complexity [38, 31].

Of interest is that both HE and DE show better performance once complexity rises. We see the potential of DE to exploit the sequence of dimensions, finding those *Bandits* which prove to be crucial in the value function and outperforming HE.

Upon inspection it is shown that in this domain, as opposed to regular game domains, most time has to be spent in the Selection Phase. This would explain why both HE and DE perform slightly worse than MCTS as relatively more time is spent in the Selection Phase due to the increased depth of the tree.

This predicts that in domains where less relative time is spent in the Selection Phase, the performance gap will dissipate.

5.2.2 Sensitivity, number of Arms

Figure 5.2 shows the sensitivity of the various algorithms over both arms and the number of bandits. This experiment is setup to show the sensitivity over both the branching factor in dimensions and in actions per dimension.

As the actions per dimension (*Arms*) increases we see that except for LSI and MCTS, the performance of all algorithms increases. For LSI we see that while performance increases over the number of *Bandits*, it falters over the number of *Arms*. This is most likely due to the fact that inherent in the algorithm is the requirement to iterate over all sub-actions per dimension. (See Figure 3.5 SideInfo(T_g) and Evaluate(C^*, T_e))

Of note is the robustness of both HE and DE against action space complexity, as both algorithms outperform all others as complexity increases. As the number of arms increases, we see that the performance gap between HE and DE diminishes. An expected result, as the importance of the sequence in dimensions is proportional to number of Bandits, not Arms. An increased branching factor per dimension in the tree prohibits DE from searching the tree to a critical depth. This trend finally results in DE showing declining performance as the action space becomes too large at 64 arms, and 48 bandits (i.e.: 48^{64} or 3.9×10^{107}). Even then, both algorithms outperform all others.

Figure 5.3 shows that as complexity of the action space increases, not only the average value by HE and DE outperforms NMC and LSI, but also the confidence interval of the action value returned is smaller. Both NMC, with ϵ-greedy as primary policy, and LSI, divide computation budget equally among the dimensions of a combined action, and thus do not optimize for the dimension which affects value the most.

5.2.3 Sensitivity, non-linearity

Figure 5.4 shows the sensitivity of the various algorithms over to the non-linearity of the reward function. In this experiment we vary the variable ϵ in the CMAB model which alters the non-linear nature of the reward function. The results show the average relative performance. This is defined as the average of $\frac{v_{i;j}}{v_{max;j}}$ over all j, where $v_{i;j}$ is the value for algorithm i on trial j. Each point in this graph was the result of 1200 trials.

In the results of this experiment we see several inflection points with regards to performance between the algorithms:

1. At $\epsilon = 0.0$ we see Hierarchical MCTS outperform Dimensional MCTS. This observation can be explained by the understanding that if no individual dimensions provide substantial more reward, or are crucial in the reward function, the added complexity of the tree depth from Dimensional MCTS will only prove a hindrance. However, as the rewards from the individual arms are heterogeneous some move ordering still exists over the dimensions. And even though the reward function is linear and thus the values returned from arms are less distinctive, the algorithm identifies and focuses on those arms which are more interesting, in turn resulting in a good relative value as compared to NMC and LSI.

2. At $0.0 < \epsilon < 0.35$ we see Dimensional MCTS outperform Hierarchical MCTS as expected. Dimensional MCTS identifies and exploits the dimensions which provide a substantial contribution to the reward function. 0.35 identifies the point in this setup where the number of interesting dimensions starts to outrank the depth at which Dimensional MCTS can explore the tree efficiently.

3. At $\epsilon > 0.5$ we see the behaviour of LSI change. As the dimensions that MCTS can explore given the budget tend to be non-linear, we see that Dimensional MCTS explores those most interesting first. This results in a more reliable value and thus outperforms Hierarchical MCTS. However, we see that LSI starts to increase in relative performance as not all non-linear dimensions are properly exploited by both MCTS variants. LSI guarantees a minimal allocation of the budget per dimension, and thus provides a baseline for each dimension.

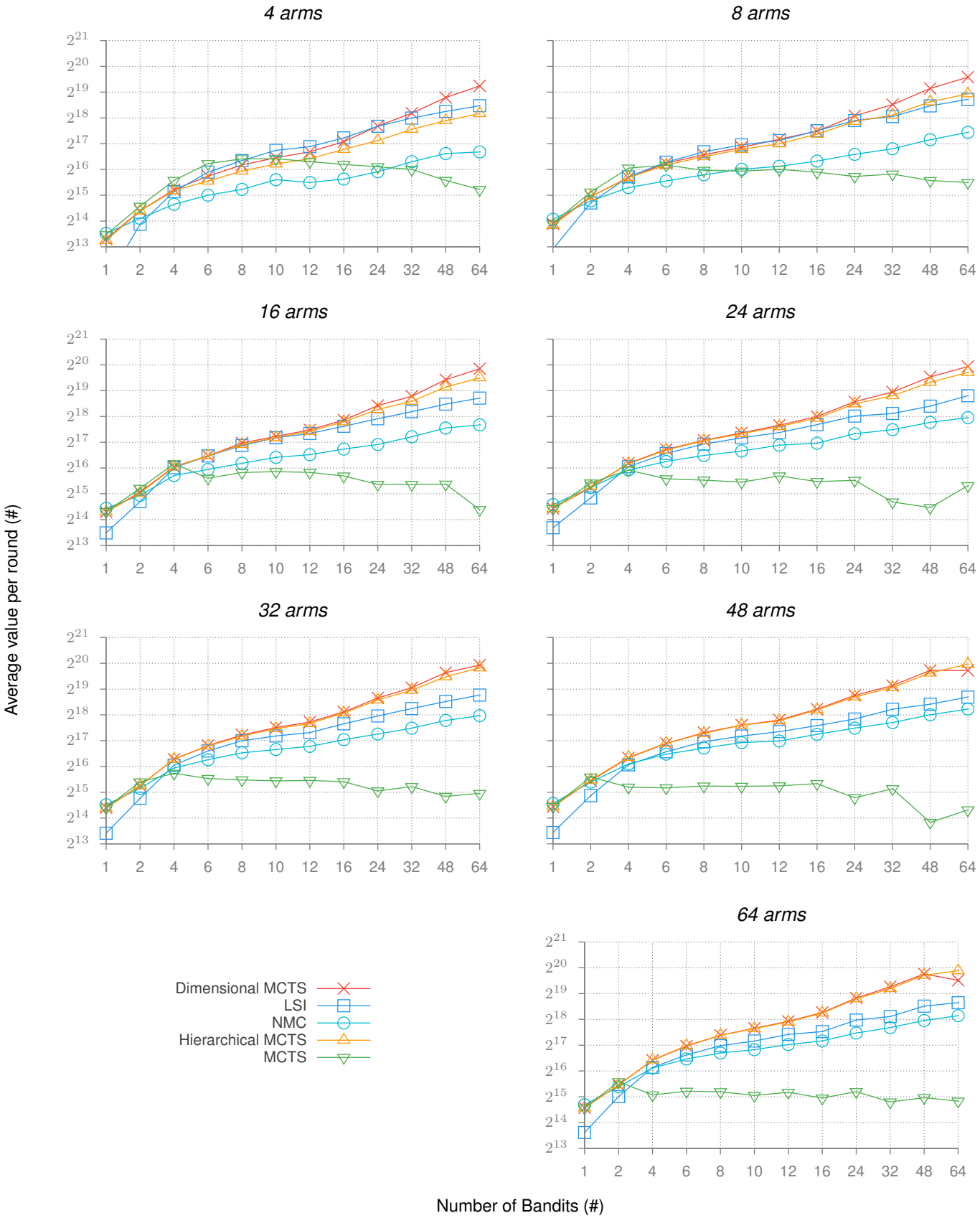

Figure 5.2: Average score of techniques over number of Bandits and Arms in action space. *4, 8, 16, 24, 32, 48 and 64 Arms per Bandit.*

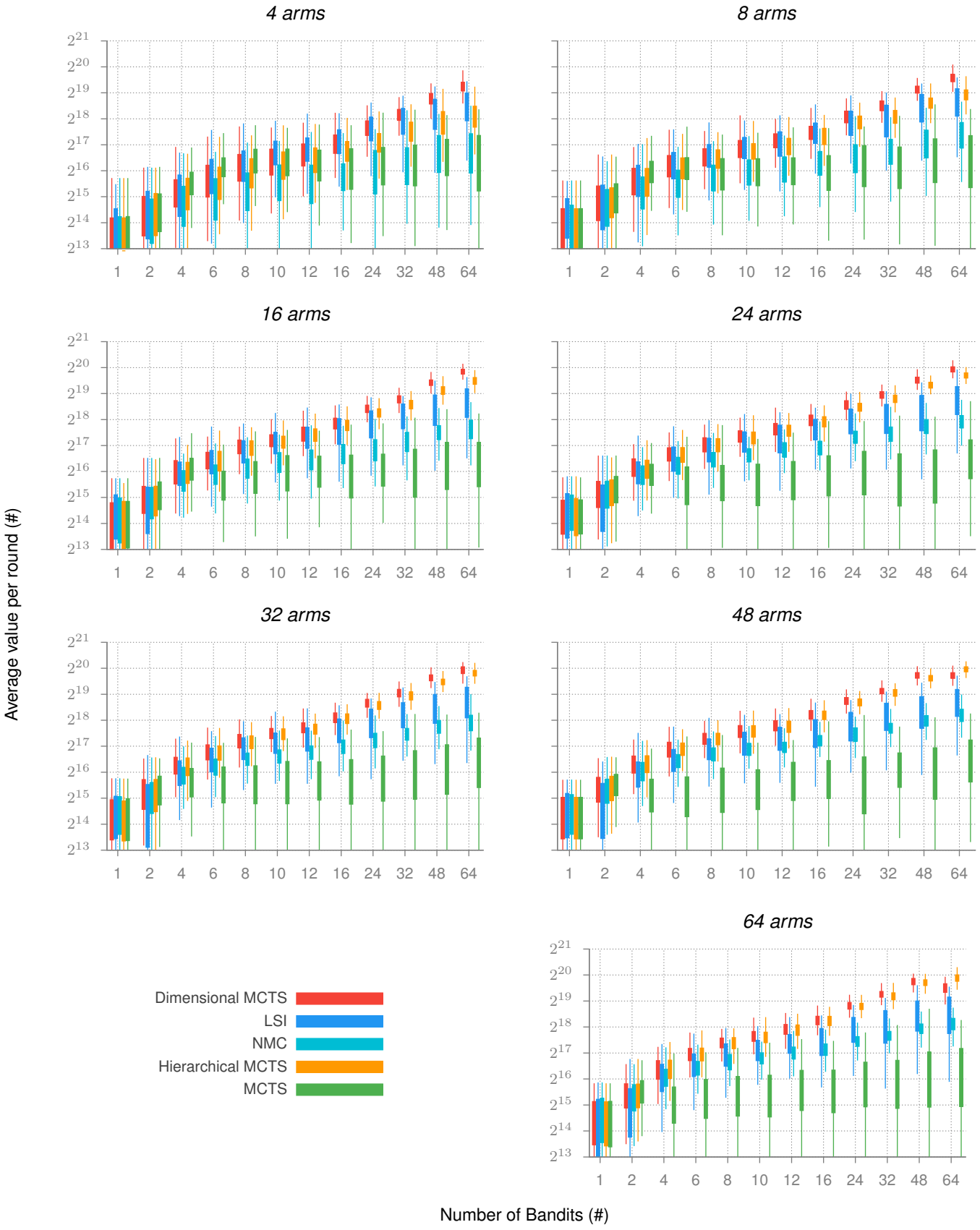

Figure 5.3: Boxplot score of techniques over number of Bandits and Arms in action space. *4, 8, 16, 24, 32, 48 and 64 Arms per Bandit.*

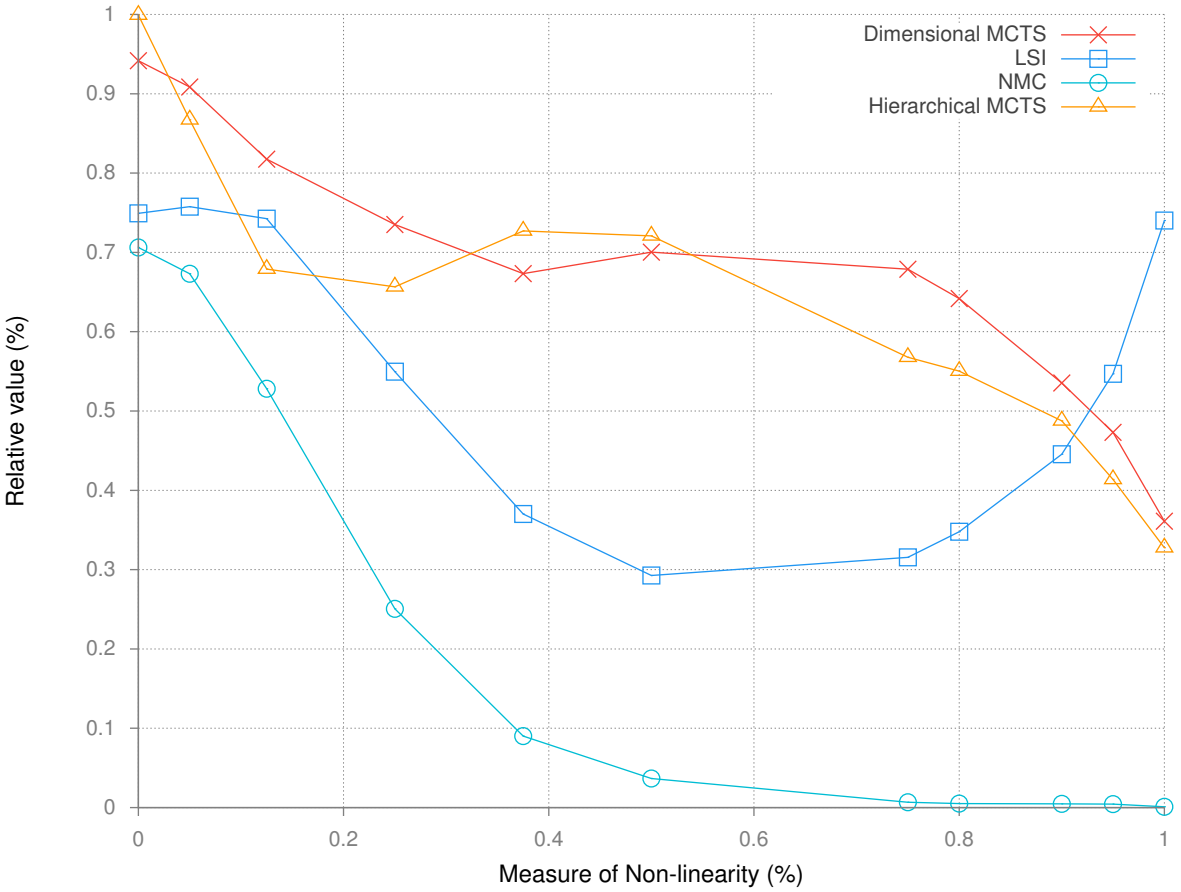

Figure 5.4: Averaged relative score with regards to best value that round. *32 Arms, 32 Bandits.*

4. At $\epsilon > 0.95$ we finally see LSI outperform both MCTS variants. As slowly every component in the reward function becomes non-linear, time must be spent on all components to prevent a collapse in value.

Apart from these specific observations we see that NMC, as expected, does not react well to the violation of the naïve assumption and performs worse as the linear components of the reward function are transformed into non-linear components.

5.3 Berlin

The following sections outline the results of the experiments performed on the domain of Berlin. Section 5.3.1 compares the performance of the enhancements to MCTS as outlined in Section 4. In Subsection 5.3.2 the performance of different playout strategies on the best performing enhancements from Subsection 4 are compared. Then in Section 5.3.3 the findings of the previous experiments are combined in a comparison of the playing strength of the various algorithms outlined in this thesis.

All experiments for Berlin are played on the *"2 Player - Standard"* map, as depicted in Figure 5.5. This map is of sufficient complexity such that we showcase the expansive action space, and used in the online AI tournament of Berlin, while maintaining a symmetrical setup so no position has an advantage. The choices made by other players are cleared before search is started for the current player, as to avoid any bias inherent in simultaneous turn games. As a performance baseline, we compare the algorithms against the Genghis Khan AI. Genghis Khan is a well-performing heuristics-based AI in the online tournament of Berlin. The standard action representation used, unless noted otherwise, is that of *node-as-agent*. Early experimentation lead to the observation that this representation is strictly better for all algorithms, except LSI. The reason for this can be explained by LSI's inherent weakness to a large sub-action branching factor, which is negated by the *soldier-as-agent* perspective. We further experiment with this in Subsection 5.3.3.

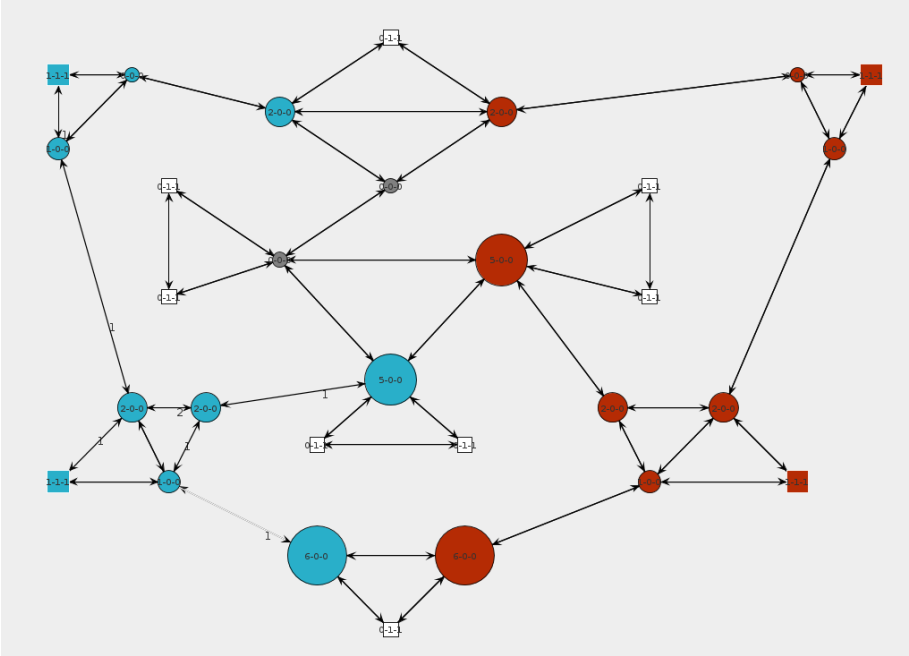

Figure 5.5: Map *"2 Player - Standard"* for Berlin. All square nodes produce a single soldier per turn and give the player one victory point at the end of the game.

For this domain a round-robin tournament is setup in which each combination of players and starting positions is tested for a preset number of matches. This rotation is introduced to remove any bias introduced by the position of the player. To ease comparison of performance, line coloring and symbol usage in graphs will be consistent for techniques between experiments.

5.3.1 Action Space Exploration

In this experiment the performance of the Hierarchical Expansion and Dimensional Expansion enhancement, and its variations, to MCTS are explored. The following variations are tested:

- *Hierarchical Expansion (HE)*, where an arbitrary ordering is picked at the start of the game.
- *Hierarchical Expansion - Random (HE, R)*, where the ordering is randomized after each expansion. This extension provides a mix between HE and DE in which the algorithm is allowed to search for a sequence over the sub-actions, without the added depth of the ply which selects the dimension. However, the algorithm has no control over which dimension is selected after an action.
- *Hierarchical Expansion - Knowledge (HE, K)*, where the ordering of dimensions explored is based on a heuristic function. In Berlin the distance of the node to the nearest enemy owned node is used. This ensures that nodes that lie along the frontier are explored first.
- *Dimensional Expansion (DE)*, the application of DE without any alterations.
- *Dimensional Expansion - Nodes (DE,N)*, a variation on DE in which an alternative action representation is used to expand over. In this alternative representation a new layer of expansion is added in which we iterate over the number of soldiers to send out over the edges. This is done in an effort to curb the branching factor when using the *node-as-agent* perspective in the action representation.

The detailed results of the experiment can be reviewed in Appendix A. In Figure 5.6 the overall performance of these techniques in the previously outlined tournament setup is shown. Figure 5.6 is primarily intended as a showcase of the overall behavior of each algorithm with regards to an increase in computation time. Figure 5.7 showcases the relative performance between algorithms better. This structure, where we explore both the behavior and relative performance, will be used throughout further experiments. From these figures the following are apparent:

- *MCTS* is outperformed by all other algorithms.
- *MCTS - HE,K* is the best performing algorithm (≥ 0.8 win rate), outperforming both variations of DE. However, as calculation time increases relative performance drops. (As compared to *MCTS - HE* and *MCTS - HE,R*)
- The importance of the inherent ordering in dimensions is shown through the performance difference between *MCTS - HE,K*, *MCTS - HE,R* and *MCTS - HE*.

From inspection of game plays it becomes apparent that while MCTS performs well in the early game, performance drops drastically as complexity of the game, and thus the branching factor, increases. As the player owns more nodes, the combinatorial nature of the action space ensures the computation budget prohibits expansion of even a single layer.

As the calculation budget is increased, the initial lead provided by domain knowledge lessens. In appendix A we see that the performance of *MCTS - HE,K* as compared to its variations drops significantly. *MCTS - HE,K* ensures that even in budget-constrained situations the search will explore nodes crucial according to the heuristics. However as the budget increases the other variations ensure exploration of these nodes as well and at the same time explore nodes incorrectly deemed worthless by heuristics.

Appendix A shows us that DE and its variations outperform uninformed HE at lower budgets, but lose relative performance once the computation budget increases. Upon playthrough inspection we see that DE correctly identifies nodes of importance and explores these first. However, Berlin is a domain with a low playout per second performance. And as such, the number of playouts in the experimental setup is not enough to overcome the added increase in tree depth, and performance falters.

DE is an expansion technique that, without extensions which provide information on actions played in other branches of the tree such as RAVE, will lose significant side-information on the performance of actions played in different branches. This loss of side-information versus depth of exploration is the trade-off that needs to be considered when applying the technique. We further explore this phenomenon in the experiment of 5.3.4 and show ways to regain this loss of information.

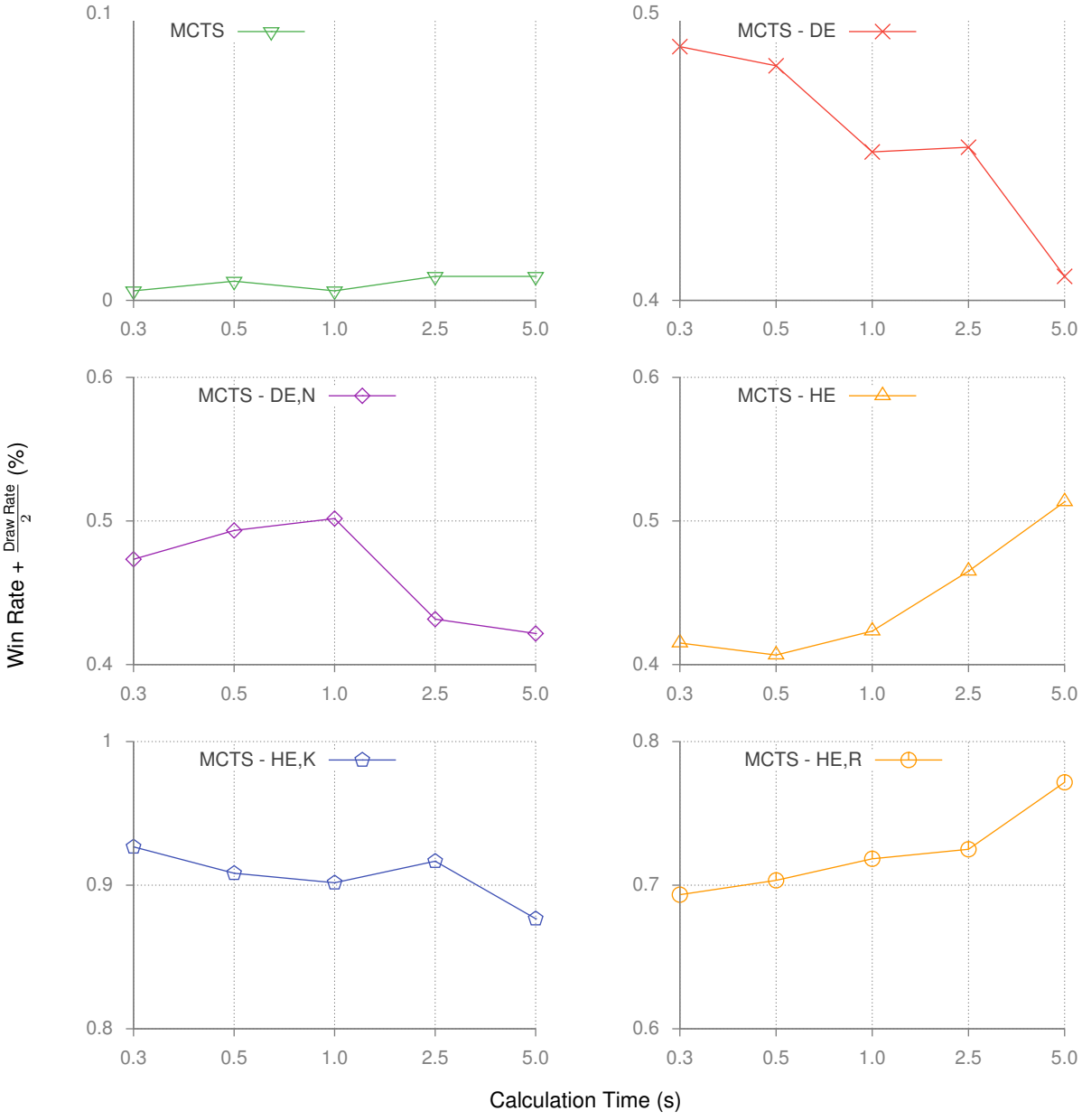

Figure 5.6: Overview of algorithm behavior in terms of overall playing strength between *expansion strategies* for MCTS, given *computation time*.

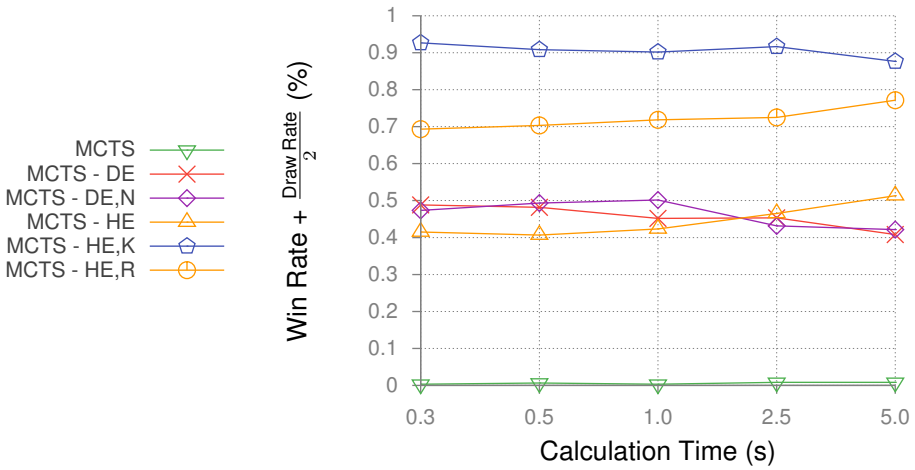

Figure 5.7: Comparison of overall playing strength behavior between *expansion strategies* for MCTS, given *computation time*.

5.3.2 Playout Policies

In this experiment the performance of various playout policies is explored. We compare the playout policies against both a pure Random agent and the Genghis Khan agent which serve as a performance baseline. As Hierarchical Expansion with domain knowledge (MCTS - HE, K) proved to be the superior technique in Section 5.3.1, this technique is used in the following experiment to find the optimal playout policy. Mixed playout strategies, in which a heuristic policy is chosen ϵ-greedy over random during playout, have been shown to outperform pure strategies depending on the domain [17, 34]. In this experiment we test the following playout policies:

- A pure heuristic playout policy (MCTS - HE, Genghis Khan), or, $\epsilon = 1.0$, where playout is delegated to the Genghis Khan Agent
- A pure random playout policy (MCTS - HE, Random), or, $\epsilon = 0.0$, where playout is delegated to the Random agent.
- The two best performing mixed strategy agents as determined by earlier experimentation, where $\epsilon = 0.25$ and $\epsilon = 0.75$. (MCTS - HE, .25 / .75)

From the results shown in Figures 5.8 and 5.9, and appendix B it becomes apparent that Genghis Khan is a formidable agent, outperforming mixed and pure strategies until a threshold computation budget is reached.

From Appendix B it becomes apparent that the Genghis Khan agent becomes particularly susceptible to a mixed strategy with heavy emphasis on heuristics as the computation budget increases. At peak computation budget, the overall win rate of MCTS - HE, .75 is 0.7 over Genghis Khan head-to-head, with a draw rate of 0.13. Of interest is also that the pure heuristic playout strategy is outperformed by both the mixed strategies and pure random playout policies at even low computation budget constraints.

This suggest that, at first, the performance cost incurred when using heuristic and thus performance intensive agents is overcome by the increase in information gain of each playout. However, these policies overfit on the problem domain by only exploring those options deemed interesting by the heuristic agent. While this is a strength when the computation budget is low, the fact that options are overlooked is a weakness that can be exploited as the computation budget increases, which is what we see in the results of this experiment. This in turn implies there exists some optimal ϵ per computation budget. These observations corroborate earlier research suggesting mixed strategies to be stronger playout policies than pure heuristic ones[17, 34].

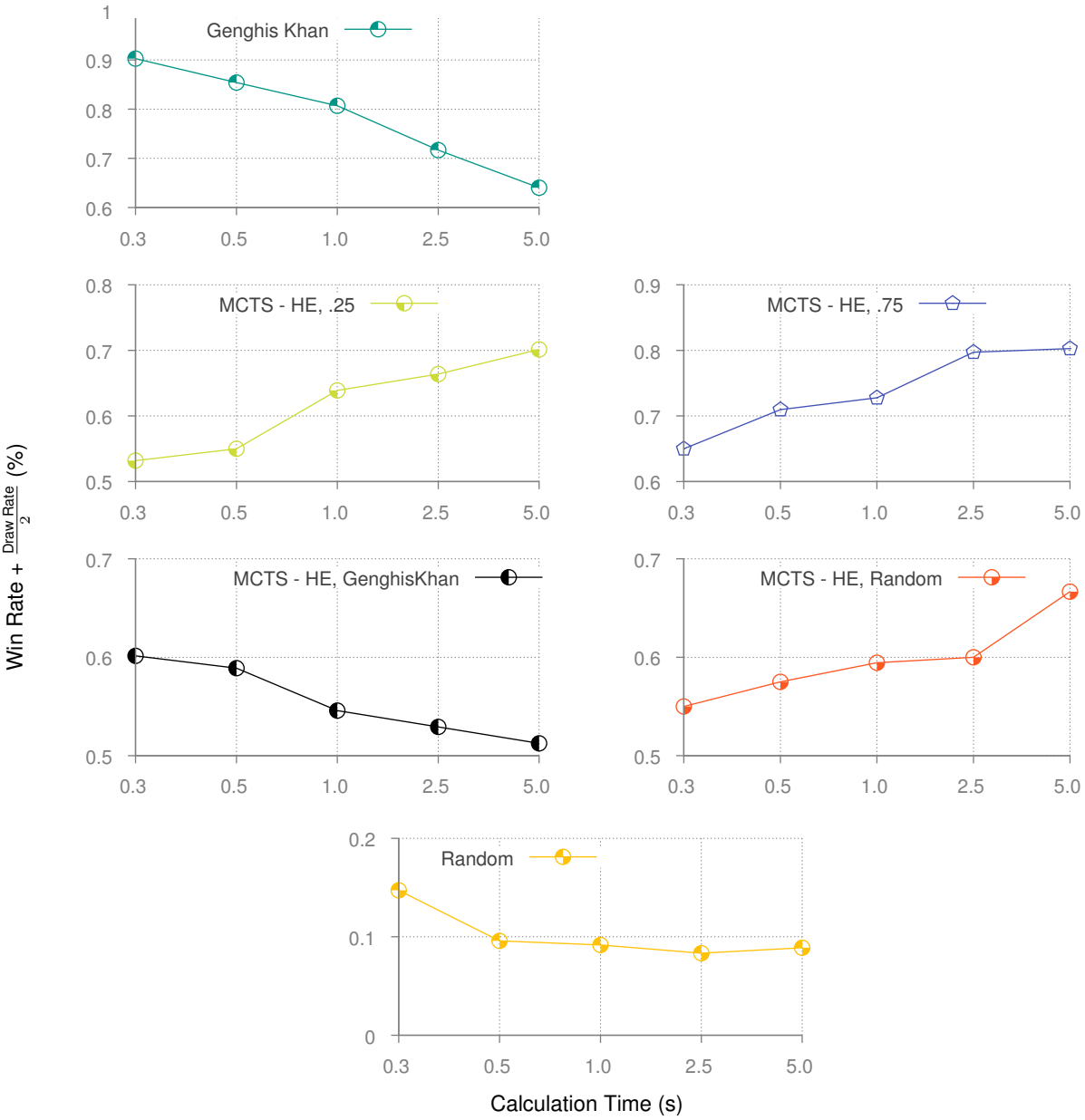

Figure 5.8: Overview of algorithm behavior in terms of playing strength behavior between *playout strategies* for MCTS, given *computation time*.

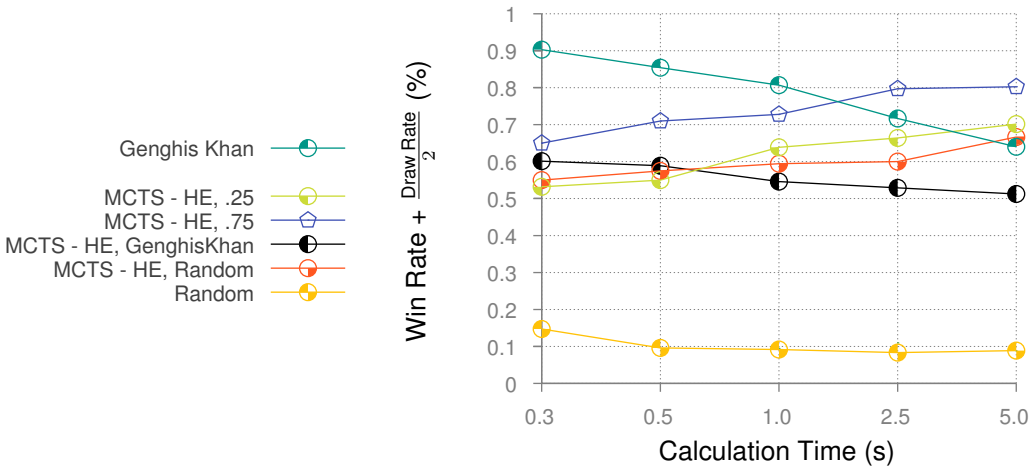

Figure 5.9: Comparison of overall playing strength behavior between *playout strategies* for MCTS, given *computation time*.

5.3.3 Search Algorithms

In the following experiment the performance of the various search algorithms outlined in this thesis are explored. As in Section 5.3.2, the performance of these algorithms is compared to both the Random and Genghis Khan agent. The playout policy for all algorithms is set to the overall best performing policy from the previous section, $\epsilon = 0.75$ with Genghis Khan.

Early experimentation lead to the observation that the alternative model representation for Berlin, in which we look at the problem from the *soldier-as-agent* perspective, lead to increased performance for LSI, but not NMC. This alternative representation is therefore included in the experimental setup as LSI^{alt}.

In this experiment the following algorithms are tested in a round-robin tournament setup:

- Time limited LSI, using the traditional action-space representation as LSI^{tl} and the alternative representation as LSI^{alt}.
- MCTS, and both the DE and HE extensions.
- NMC

From appendix C and Figures 5.10 and 5.11 it becomes apparent that MCTS is outperformed by all other algorithms, and even beaten by the Random Agent. While surprising at first, this can be explained by the fact that the Random agent samples the entire action space, while MCTS fails to as the action space increases in size during the match and thus the action space becomes innumerable given the computation budget. MCTS then returns a random decision on the limited action space it did explore, as none of the actions were sampled enough to provide a clear distinction on the best move available.

As expected, we see both heuristic agents (Random and Genghis Khan) decrease in performance as the computation budget increases. Of interest is that both variations of LSI seem to be outperformed by NMC. The results shown in Section 5.2 imply that as complexity of the CMAB increases, the performance gap between LSI and NMC decreases. The results of Section 5.2 also indicate that the performance of LSI is negatively correlated to an increase in the number of Arms in per Bandit in a CMAB model. It is therefore of the author's opinion that we are seeing the logical end conclusion of applying LSI in a complex domain with an unfavorable model representation. This is corroborated by the observation that the LSI variation using the alternative model representation, where the number of Arms per Bandit is decreased in favor of an increase in the number of Bandits, outperforms the LSI variation using the standard representation.

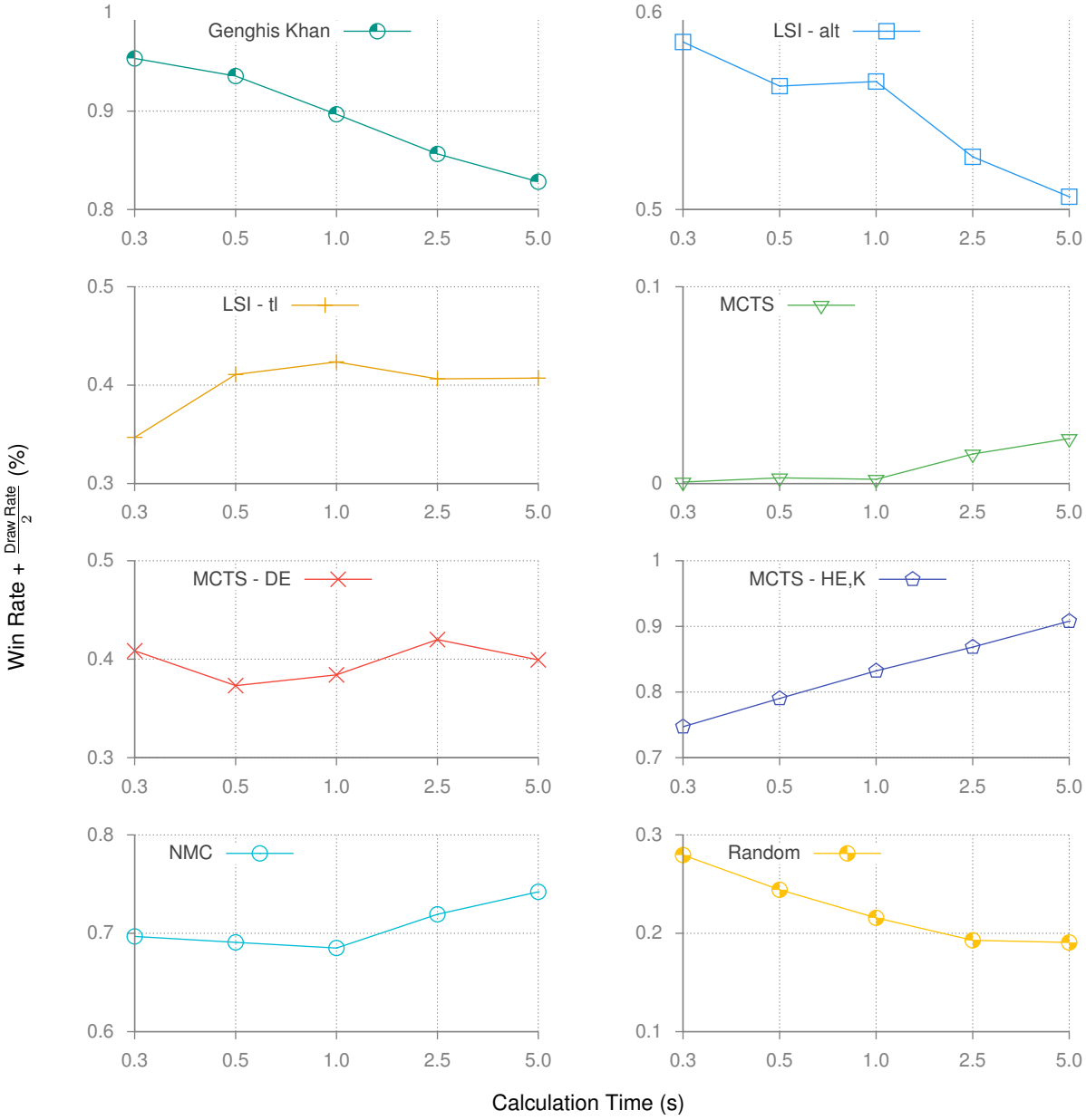

Figure 5.10: Overview of algorithm behavior in terms of overall playing strength behavior between *search techniques*, given *computation time*.

Finally, we see that HE outperforms all other algorithms, and its relative performance increases as the computation budget increases. DE suffers from the same weaknesses as outlined in Subsection 5.3.1, barely outperforming LSI, but handily beaten by NMC.

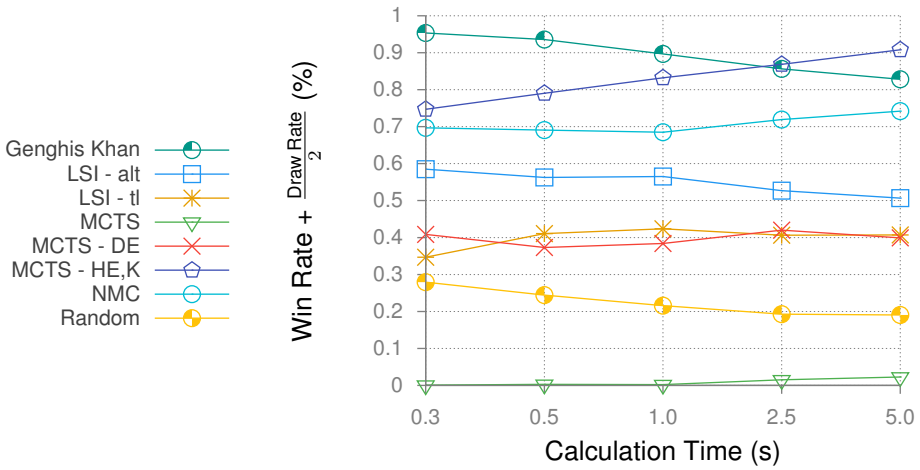

Figure 5.11: Comparison of overall playing strength behavior between *search techniques*, given *computation time*.

5.3.4 Partial Move Completion

In this section we compare the different strategies of completing a partial move returned by the search. We compare both DE and HE in an effort to show the difference in depth of exploration versus exploitation. The following strategies are compared:

- Random move completion. Any undecided node is filled with a random distribution of soldiers along edges.
- Heuristic move completion. Any undecided node is decided by the partial move provided by the Genghis Khan agent.
- RAVE move completion. Any undecided node is decided by the best performing move as returned by RAVE. RAVE in this experiment is only used for completing the partial move, not to aid the tree search policy during search.

Figure 5.12, 5.13 and appendix D show the results of these experiments. As expected, heuristic based completion of the partial move provides a significant advantage over both random and RAVE based completion at low computation budget constraints. Also, we see that both non-heuristic based policies improve over time as the computation budget increases.

RAVE provides a significant boost to all algorithms, giving a trend that seems to indicate it will outperform heuristic completion as time increases. This observation stems from the fact that both in DE and HE side-information is lost on moves played in different branches of the algorithm. We expect DE to gain the most from this approach as the loss of side-information is the largest. This is shown in the results of the experiment as DE benefits immensely from the information regained through RAVE.

Of interest is that the heuristic based approach for DE outperforms all other approaches when the calculation budget is kept below 5 seconds. This confirms the intuition that DE successfully explores and exploits those moves crucial in the game, and shows that its lack of performance in the heads-up tournament is primarily explained by its lacking search depth.

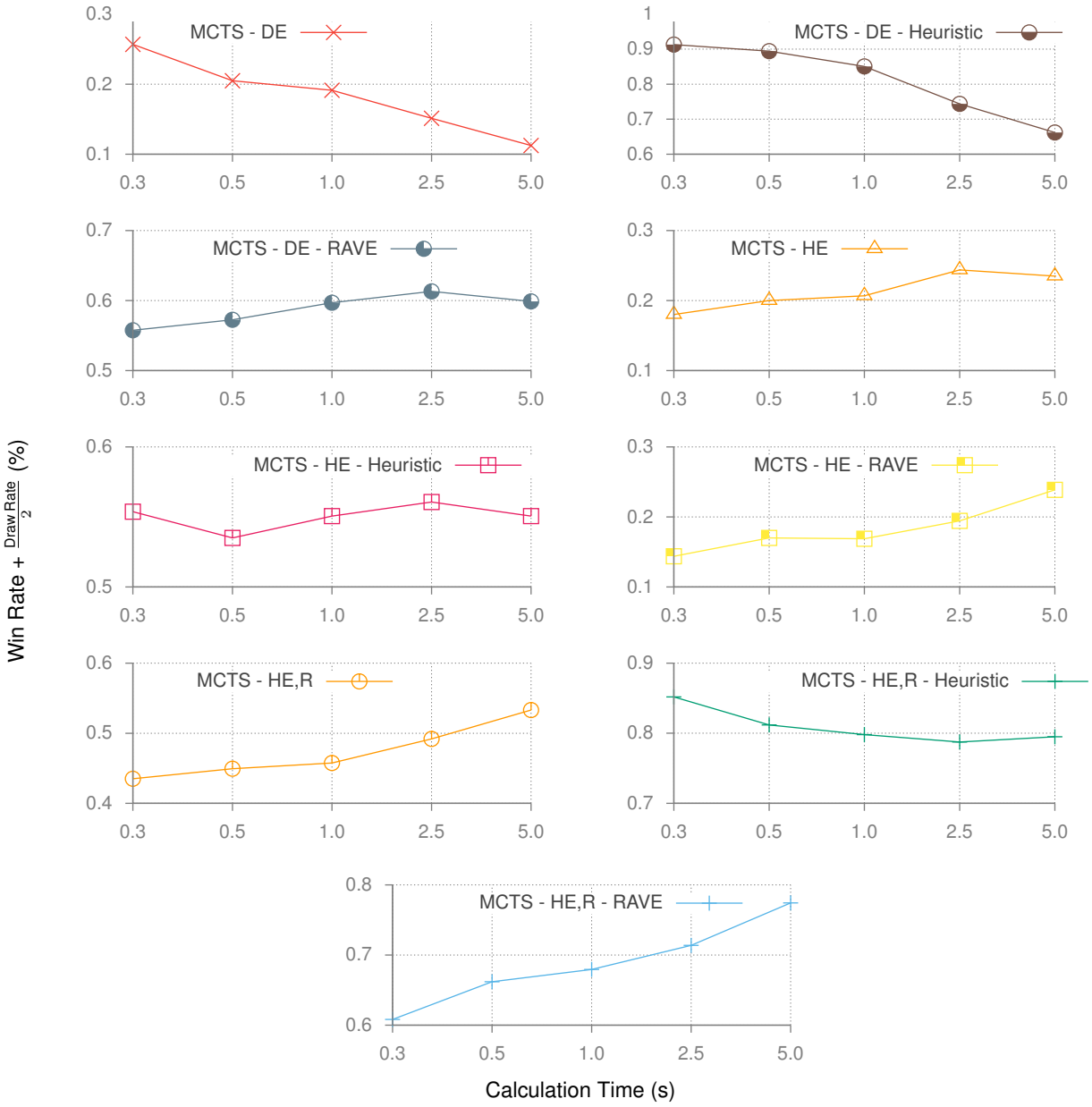

Figure 5.12: Overview of algorithm behavior in terms of overall playing strength behavior between *partial move completion* policies, given *computation time*.

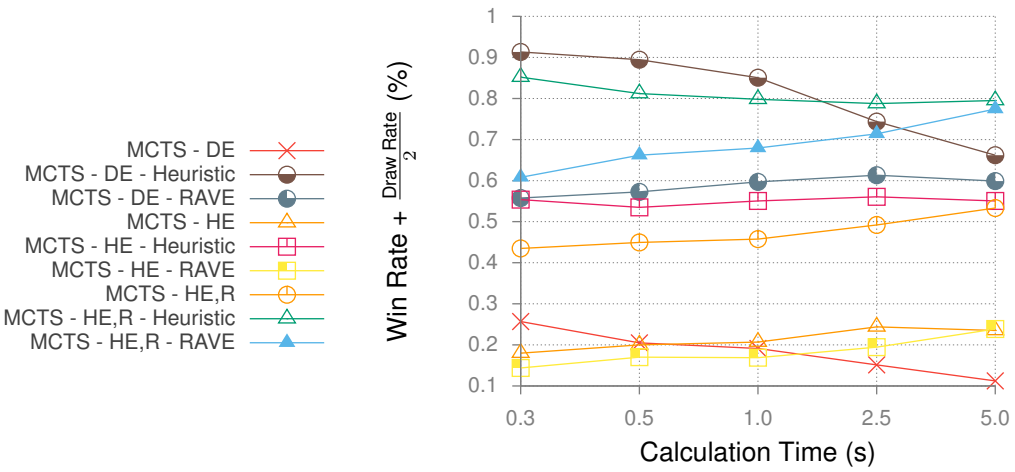

Figure 5.13: Comparison of overall playing strength behavior between *partial move completion* policies, given *computation time*.

6

Conclusion

Chapter overview: *This chapter provides a reflection on the problem statement and its subsequent research questions by reviewing the results of the experiments and drawing conclusions based on them. Finally, directions for future research are provided.*

6.1 Research Questions

1. *How can MCTS be extended to cope with a combinatorial action space?*

Chapter 1 shows that examples of research into other domains [32, 47] has shown that alternative representations of a combinatorial action space have resulted in increased performance. Naïve Monte Carlo (NMC) [38] and Linear Side Information (LSI) [31] have both been shown to perform well with the CMAB model and outperformed naïve applications of MCTS. Both approaches give inspiration to the formalization and proposal of two algorithms, Hierarchical Expansion (HE) and Dimensional Expansion (DE), in Chapter 4 which simultaneously allow MCTS to tackle action spaces of a combinatorial nature, and explore and exploit the inherent ordering over dimensions of sub-actions in a CMAB.

2. *How can MCTS be extended to exploit the side-information inherent in the CMAB model?*

Both HE and DE focus on the exploitation of the ordering of dimensions within a CMAB, but do not explicitly use the side-information inherent in the sub-actions of dimensions. However, experiment 5.3.4 shows that existing techniques that provide information over recurring actions, such as RAVE, can be used to provide side information in strategies used to complete partial moves with substantial benefit to performance. Exploratory experiments showed performance improvements when using RAVE in the search, however this remains the topic of future research and these results are not handled in this thesis.

3. *How does performance of the applied algorithms vary across variations in the CMAB model?*

Experiments on the CMAB model, as outlined in Section 5.2 show that the various algorithms react differently to changes in characteristics of the model. While the naïve application of MCTS outperforms the other algorithms at trivially sized problems, as the domain reaches any reasonable level of complexity the performance of MCTS drops. The general observation is that LSI outperforms NMC, however as the problem complexity increases, this gap lessens and is even shown to revert in the domain of Berlin. LSI seems to be particularly vulnerable to the size of the sub-action space. Both DE and HE are shown to be resistant to a rise in complexity, outperforming the other algorithms in the CMAB experimental setup. DE and HE both scale better given increasing time constraints, which is a logical outcome given that both algorithms are based on MCTS as opposed to the selection policies which optimize simple regret used by NMC and LSI. The end result being that the proposed MCTS extensions outperform LSI and NMC as the computation budget increases.

4. *What is the performance of the applied algorithms in the domain of Berlin?*

The experiments in Section 5.3 show that the exploitation of the priority in dimensions through HE outperforms both NMC and LSI, and beats the heuristic agent Genghis Khan for higher time settings. With the application of RAVE on the partial move completion we see that the requirement for heuristic knowledge in domain ordering can be removed while maintaining and even improving performance. While HE with heuristic knowledge outperforms DE, without reasonable heuristics it is outperformed by DE when RAVE is used to augment partial moves. This observation is particularly interesting to General Game Playing (GGP), as no heuristics are known on the game beforehand. The improvements on performance gained through RAVE in the partial playout policies seem to indicate that the performance of DE could be further improved if the side-information can be used within the search. Of note is that the ordering of dimensions heavily influences the performance of HE, following from the observation that a stochastic ordering reset after every expansion performs better than an arbitrarily picked ordering. This variation of HE (HE,R), provides a mix between DE and HE in the sense that the algorithm does search for a sequence over the sub-actions, but has no control over this exploration as the dimension is determined randomly after each expansion. The benefit in contrast to DE is that no ply is added which determines the dimension to explore.

6.2 Problem Statement

Determine the performance of state-of-the-art techniques in domains which can be modeled as a Combinatorial Multi-Armed Bandit (CMAB), and formulate extensions to MCTS which improve its performance in this domain.

In this thesis two alternative representations of the action space for a CMAB are presented which exploit the ordering of dimensions and allow MCTS to efficiently search these domains. The basis for these techniques lies in the informal application of these strategies in previous research and the success of NMC and LSI in the domain of μRTS over MCTS. An alternative definition of LSI is proposed which does not rely on a simulation budget, handles a more expansive sub-action space and performs equal to the original algorithm.

Results show that the techniques proposed, HE and DE, outperform naïve MCTS, NMC and LSI in various experimental setups in a straight-forward adaption of the CMAB model, and HE outperforms NMC, LSI and DE in the domain of Berlin. However, the decreased performance in Berlin for DE can be compensated by extending the algorithm with proper partial move completion. It is shown that both LSI and NMC outperform both HE and DE in situations with a low computational budget, but this performance gap lessens as the budget increases. NMC outperforms LSI in the domain of Berlin, continuing the observation first shown in the CMAB model that LSI suffers in performance as the complexity of the sub-action space increases. The effectiveness of exploring and exploiting the ordering of dimensions is shown through the effective playing strength of both HE and DE and its variations.

6.3 Future Research

While it was shown that there is performance to be gained through exploitation of the ordering in dimensions in a CMAB, no experiments have been performed yet on how side-information could be exploited given the new representations using existing techniques. Research has shown significant gains in performance through the use of selection policies in UCT which are specifically targeted at domains with a large branching factor [9]. These techniques have not been discussed in this thesis because they cannot handle an innumerable action space and, more importantly, ignore the inherent side-information in the domain. However, through HE and DE, these techniques become viable. The exploitation of side-information could then be exploited through the use of different extensions to MCTS such as RAVE. The performance gap between MCTS, NMC and LSI in the domain of Berlin is of interest and could be explained through the different exploration strategies used by the algorithms.

Bibliography

[1] Selim G. Akl and Monroe M. Newborn. "The principal continuation and the killer heuristic". In: *Proceedings of the 1977 annual conference*. ACM. 1977, pp. 466–473.

[2] L. Victor Allis, Maarten van der Meulen, and H. Jaap Van Den Herik. "Proof-number search". In: *Artificial Intelligence* 66.1 (1994), pp. 91–124.

[3] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. "Monte carlo tree search in hex". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 2.4 (2010), pp. 251–258.

[4] Jean-Yves Audibert and Sébastien Bubeck. "Best arm identification in multi-armed bandits". In: *COLT-23th Conference on Learning Theory-2010*. 2010, p. 13.

[5] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[6] Radha-Krishna Balla and Alan Fern. "UCT for Tactical Assault Planning in Real-Time Strategy Games." In: *IJCAI*. 2009, pp. 40–45.

[7] *Berlin AI Competition*. URL: `http://berlin-ai.com/`.

[8] 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". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 4.1 (2012), pp. 1–43.

[9] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. "Pure exploration in finitely-armed and continuous-armed bandits". In: *Theoretical Computer Science* 412.19 (2011), pp. 1832–1852.

[10] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. "Deep Blue". In: *Artificial intelligence* 134.1 (2002), pp. 57–83.

[11] Tristan Cazenave. "Sequential halving applied to trees". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 7.1 (2015), pp. 102–105.

[12] G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H. van den Herik, and Bruno Bouzy. "Progressive strategies for monte-carlo tree search". In: *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*. World of Science. 2007, pp. 655–661.

[13] Wei Chen, Yajun Wang, and Yang Yuan. "Combinatorial multi-armed bandit: General framework and applications". In: *Proceedings of the 30th International Conference on Machine Learning*. 2013, pp. 151–159.

[14] Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. "Transpositions and move groups in Monte Carlo tree search". In: *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE. 2008, pp. 389–395.

[15] Michael Chung, Michael Buro, and Jonathan Schaeffer. "Monte-Carlo planning in RTS games". In: *CIG*. 2005.

[16] Rémi Coulom. "Efficient selectivity and backup operators in Monte-Carlo tree search". In: *Computers and games*. Springer, 2007, pp. 72–83.

[17] Niek G.P. Den Teuling and Mark H.M. Winands. "Monte-Carlo Tree Search for the simultaneous move game Tron". In: *Computer Games Workshop at ECAI 2012, pp. 126-141. Montpellier, France* (2012).

[18] Timo Ewalds. "Playing and solving Havannah". In: *M.sc Thesis, University of Alberta* (2012).

[19] David Fotland. "Building a world-champion arimaa program". In: *Computers and Games*. Springer, 2006, pp. 175–186.

[20] Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. "Learning multiuser channel allocations in cognitive radio networks: A combinatorial multi-armed bandit formulation". In: *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*. IEEE. 2010, pp. 1–9.

[21] Xiaocong Gan, Yun Bao, and Zhangang Han. "Real-time Search method in non-deterministic game - Ms. Pacman". In: *ICGA JOURNAL* 34.4 (2011), pp. 209–222.

[22] Sylvain Gelly and David Silver. "Achieving Master Level Play in 9 x 9 Computer Go." In: *AAAI*. Vol. 8. 2008, pp. 1537–1540.

[23] Sylvain Gelly and David Silver. "Combining online and offline knowledge in UCT". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 273–280.

[24] Sylvain Gelly and David Silver. "Monte-Carlo tree search and rapid action value estimation in computer Go". In: *Artificial Intelligence* 175.11 (2011), pp. 1856–1875.

[25] *Genghis Khan Agent*. URL: `https://github.com/mathieugagne/genghis-khan`.

[26] T. Goslin and P. Andruszkiewicz. *Monte-Carlo Tree Search in TOTAL WAR: ROME II*. Aug. 2014. URL: `http://aigamedev.com/open/coverage/mcts-rome-ii/`.

[27] Zohar Karnin, Tomer Koren, and Oren Somekh. "Almost optimal exploration in multi-armed bandits". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 1238–1246.

[28] Donald E. Knuth and Ronald W. Moore. "An analysis of alpha-beta pruning". In: *Artificial intelligence* 6.4 (1979), pp. 293–326.

[29] Levente Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.

[30] Sven Koenig. "Agent-centered search". In: *AI Magazine* 22.4 (2001), p. 109.

[31] Antonín Komenda, Alexander Shleyfman, and Carmel Domshlak. "On Robustness of CMAB Algorithms: Experimental Approach". In: *Computer Games*. Springer, 2014, pp. 16–28.

[32] Tomas Kozelek. "Methods of MCTS and the game Arimaa". In: *M.sc Thesis, Charles University, Prague, Faculty of Mathematics and Physics* (2009).

[33] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in Applied Mathematics* 6.1 (1985), pp. 4–22.

[34] Dale LaPlante and Christopher Nota. "Improvements to MCTS Simulation Policies in Go". In: *M.Sc. Thesis* (2014).

[35] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. "The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 1.1 (2009), pp. 73–89.

[36] Richard J Lorentz. "Amazons discover monte-carlo". In: *Computers and games*. Springer, 2008, pp. 13–24.

[37] T. Anthony Marsland. "A review of game-tree pruning". In: *ICCA journal* 9.1 (1986), pp. 3–19.

[38] Santiago Ontanón. "The combinatorial multi-armed bandit problem and its application to real-time strategy games". In: *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2013.

[39] Daniel R. Page. "Generalized algorithm for restricted weak composition generation". In: *Journal of Mathematical Modelling and Algorithms in Operations Research* 12.4 (2013), pp. 345–372.

[40] Tom Pepels, Tristan Cazenave, Mark H.M. Winands, and Marc Lanctot. "Minimizing Simple and Cumulative Regret in Monte-Carlo Tree Search". In: *Computer Games*. Springer, 2014, pp. 1–15.

[41] Marc Ponsen, Steven De Jong, and Marc Lanctot. "Computing approximate nash equilibria and robust best-responses using sampling". In: *Journal of Artificial Intelligence Research* (2011), pp. 575–605.

[42] Alexander Reinefeld. “An improvement of the Scout tree-search algorithm”. In: *ICCA Journal* 6.4 (1983), pp. 4–14.

[43] Arpad Rimmel, Olivier Teytaud, Chang-Shing Lee, Shi-Jim Yen, Mei-Hui Wang, and Shang-Rong Tsai. “Current frontiers in computer Go”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 2.4 (2010), pp. 229–238.

[44] G.J.B. Roelofs. “Monte Carlo Tree Search in a modern board game framework”. In: *B.sc. Thesis, Maastricht University* (2012).

[45] Jahn-Takeshi Saito, Mark H.M. Winands, Jos W.H.M. Uiterwijk, and H. Jaap van den Herik. “Grouping nodes for Monte-Carlo tree search”. In: *Computer Games Workshop*. 2007, pp. 276–283.

[46] Jonathan Schaeffer. “The history heuristic and alpha-beta search enhancements in practice”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 11.11 (1989), pp. 1203–1212.

[47] Dennis Soemers. “Tactical Planning Using MCTS in the Game of StarCraft”. In: *B.sc. Thesis, Maastricht University* (2014).

[48] Omar Syed and Aamir Syed. “Arimaa-a new game designed to be difficult for computers”. In: *ICGA Journal* 26.2 (2003), pp. 138–139.

[49] István Szita, Guillaume Chaslot, and Pieter Spronck. “Monte-carlo tree search in Settlers of Catan”. In: *Advances in Computer Games*. Springer, 2010, pp. 21–32.

[50] Mandy J.W. Tak, Marc Lanctot, and Mark H.M. Winands. “Monte Carlo Tree Search variants for simultaneous move games”. In: *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE. 2014, pp. 232–239.

[51] *Thirdside Development*. URL: `http://thirdside.ca/`.

[52] Alan M. Turing. “Chess”. In: *Reprinted in 1988 in Computer Chess Compendium*. Springer, 1953, pp. 14–17.

[53] Gabriel Van Eyck and Martin Müller. “Revisiting move groups in monte-carlo tree search”. In: *Advances in Computer Games*. Springer, 2012, pp. 13–23.

[54] Peter Whittle. “Multi-armed bandits and the Gittins index”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1980), pp. 143–149.

[55] Haizhi Zhong. “Building a strong Arimaa-playing program”. PhD thesis. University of Alberta, 2005.

Appendices

Appendix A

Experimental Results: Berlin - Expansion

The following chapter contains the experimental results for the comparison of various expansion policies in the domain of Berlin. Each table shows the win ratio (W) and draw ratio (D) per match up. If the row algorithm outperforms the column algorithm, defined as $W + \frac{D}{2} \geq 0.5$, then the entry is marked as bold. The experimental setup of each datapoint is **60** matches, equally divided over all combinations for starting positions. The matches are played on the *2 Player - Standard* map.

Row *vs* Column	**MCTS**	**MCTS DE**	**MCTS DE,N**	**MCTS HE**	**MCTS HE,K**	**MCTS HE,R**	*Total*
MCTS		$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.02}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$
MCTS - DE	$\mathbf{W_{1.00}/D_{0.00}}$		$\mathbf{W_{0.45}/D_{0.13}}$	$\mathbf{W_{0.60}/D_{0.15}}$	$W_{0.02}/D_{0.03}$	$W_{0.17}/D_{0.10}$	$W_{0.45}/D_{0.08}$
MCTS - DE,N	$\mathbf{W_{1.00}/D_{0.00}}$	$W_{0.42}/D_{0.13}$		$\mathbf{W_{0.48}/D_{0.15}}$	$W_{0.02}/D_{0.08}$	$W_{0.17}/D_{0.20}$	$W_{0.42}/D_{0.11}$
MCTS - HE	$\mathbf{W_{0.98}/D_{0.00}}$	$W_{0.25}/D_{0.15}$	$W_{0.37}/D_{0.15}$		$W_{0.08}/D_{0.03}$	$W_{0.12}/D_{0.22}$	$W_{0.36}/D_{0.11}$
MCTS - HE,K	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.95}/D_{0.03}}$	$\mathbf{W_{0.90}/D_{0.08}}$	$\mathbf{W_{0.88}/D_{0.03}}$		$\mathbf{W_{0.72}/D_{0.22}}$	$\mathbf{W_{0.89}/D_{0.07}}$
MCTS - HE,R	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.73}/D_{0.10}}$	$\mathbf{W_{0.63}/D_{0.20}}$	$\mathbf{W_{0.67}/D_{0.22}}$	$W_{0.07}/D_{0.22}$		$\mathbf{W_{0.62}/D_{0.15}}$

Figure A.1: Results for experiment *Expansion*, given 250ms.

Row *vs* Column	**MCTS**	**MCTS DE**	**MCTS DE,N**	**MCTS HE**	**MCTS HE,K**	**MCTS HE,R**	*Total*
MCTS		$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.02}/D_{0.00}$	$W_{0.00}/D_{0.02}$	$W_{0.00}/D_{0.02}$	$W_{0.00}/D_{0.01}$
MCTS - DE	$\mathbf{W_{1.00}/D_{0.00}}$		$W_{0.37}/D_{0.22}$	$\mathbf{W_{0.53}/D_{0.15}}$	$W_{0.02}/D_{0.07}$	$W_{0.17}/D_{0.22}$	$W_{0.42}/D_{0.13}$
MCTS - DE,N	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.42}/D_{0.22}}$		$\mathbf{W_{0.52}/D_{0.15}}$	$W_{0.05}/D_{0.08}$	$W_{0.17}/D_{0.18}$	$W_{0.43}/D_{0.13}$
MCTS - HE	$\mathbf{W_{0.98}/D_{0.00}}$	$W_{0.32}/D_{0.15}$	$W_{0.33}/D_{0.15}$		$W_{0.02}/D_{0.05}$	$W_{0.12}/D_{0.18}$	$W_{0.35}/D_{0.11}$
MCTS - HE,K	$\mathbf{W_{0.98}/D_{0.02}}$	$\mathbf{W_{0.92}/D_{0.07}}$	$\mathbf{W_{0.87}/D_{0.08}}$	$\mathbf{W_{0.93}/D_{0.05}}$		$\mathbf{W_{0.63}/D_{0.20}}$	$\mathbf{W_{0.87}/D_{0.08}}$
MCTS - HE,R	$\mathbf{W_{0.98}/D_{0.02}}$	$\mathbf{W_{0.62}/D_{0.22}}$	$\mathbf{W_{0.65}/D_{0.18}}$	$\mathbf{W_{0.70}/D_{0.18}}$	$W_{0.17}/D_{0.20}$		$\mathbf{W_{0.62}/D_{0.16}}$

Figure A.2: Results for experiment *Expansion*, given 500ms.

Row *vs* Column	**MCTS**	**MCTS DE**	**MCTS DE,N**	**MCTS HE**	**MCTS HE,K**	**MCTS HE,R**	*Total*
MCTS		$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.02}/D_{0.00}$	$W_{0.00}/D_{0.00}$
MCTS - DE	$\mathbf{W_{1.00}/D_{0.00}}$		$W_{0.35}/D_{0.20}$	$\mathbf{W_{0.45}/D_{0.15}}$	$W_{0.00}/D_{0.08}$	$W_{0.18}/D_{0.12}$	$W_{0.40}/D_{0.11}$
MCTS - DE,N	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.45}/D_{0.20}}$		$\mathbf{W_{0.52}/D_{0.20}}$	$W_{0.03}/D_{0.17}$	$W_{0.17}/D_{0.12}$	$\mathbf{W_{0.43}/D_{0.14}}$
MCTS - HE	$\mathbf{W_{1.00}/D_{0.00}}$	$W_{0.40}/D_{0.15}$	$W_{0.28}/D_{0.20}$		$W_{0.02}/D_{0.08}$	$W_{0.13}/D_{0.13}$	$W_{0.37}/D_{0.11}$
MCTS - HE,K	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.92}/D_{0.08}}$	$\mathbf{W_{0.80}/D_{0.17}}$	$\mathbf{W_{0.90}/D_{0.08}}$		$\mathbf{W_{0.57}/D_{0.32}}$	$\mathbf{W_{0.84}/D_{0.13}}$
MCTS - HE,R	$\mathbf{W_{0.98}/D_{0.00}}$	$\mathbf{W_{0.70}/D_{0.12}}$	$\mathbf{W_{0.72}/D_{0.12}}$	$\mathbf{W_{0.73}/D_{0.13}}$	$W_{0.12}/D_{0.32}$		$\mathbf{W_{0.65}/D_{0.14}}$

Figure A.3: Results for experiment *Expansion*, given 1000ms.

Row *vs* Column	**MCTS**	**MCTS DE**	**MCTS DE,N**	**MCTS HE**	**MCTS HE,K**	**MCTS HE,R**	*Total*
MCTS		$W_{0.00}/D_{0.02}$	$W_{0.00}/D_{0.00}$	$W_{0.02}/D_{0.02}$	$W_{0.00}/D_{0.02}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.01}$
MCTS - DE	$\mathbf{W_{0.98}/D_{0.02}}$		$W_{0.45}/D_{0.08}$	$\mathbf{W_{0.42}/D_{0.25}}$	$W_{0.02}/D_{0.08}$	$W_{0.13}/D_{0.10}$	$W_{0.40}/D_{0.11}$
MCTS - DE,N	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.47}/D_{0.08}}$		$W_{0.23}/D_{0.33}$	$W_{0.02}/D_{0.10}$	$W_{0.07}/D_{0.23}$	$W_{0.36}/D_{0.15}$
MCTS - HE	$\mathbf{W_{0.97}/D_{0.02}}$	$W_{0.33}/D_{0.25}$	$\mathbf{W_{0.43}/D_{0.33}}$		$W_{0.05}/D_{0.03}$	$W_{0.08}/D_{0.28}$	$W_{0.37}/D_{0.18}$
MCTS - HE,K	$\mathbf{W_{0.98}/D_{0.02}}$	$\mathbf{W_{0.90}/D_{0.08}}$	$\mathbf{W_{0.88}/D_{0.10}}$	$\mathbf{W_{0.92}/D_{0.03}}$		$\mathbf{W_{0.65}/D_{0.27}}$	$\mathbf{W_{0.87}/D_{0.10}}$
MCTS - HE,R	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.77}/D_{0.10}}$	$\mathbf{W_{0.70}/D_{0.23}}$	$\mathbf{W_{0.63}/D_{0.28}}$	$W_{0.08}/D_{0.27}$		$\mathbf{W_{0.64}/D_{0.18}}$

Figure A.4: Results for experiment *Expansion*, given 2500ms.

Row *vs* Column	**MCTS**	**MCTS DE**	**MCTS DE,N**	**MCTS HE**	**MCTS HE,K**	**MCTS HE,R**	*Total*
MCTS		$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.00}$	$W_{0.00}/D_{0.03}$	$W_{0.00}/D_{0.00}$	$W_{0.02}/D_{0.02}$	$W_{0.00}/D_{0.01}$
MCTS - DE	$\mathbf{W_{1.00}/D_{0.00}}$		$W_{0.43}/D_{0.08}$	$W_{0.23}/D_{0.25}$	$W_{0.03}/D_{0.07}$	$W_{0.07}/D_{0.15}$	$W_{0.35}/D_{0.11}$
MCTS - DE,N	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.48}/D_{0.08}}$		$W_{0.22}/D_{0.25}$	$W_{0.07}/D_{0.05}$	$W_{0.08}/D_{0.13}$	$W_{0.37}/D_{0.10}$
MCTS - HE	$\mathbf{W_{0.97}/D_{0.03}}$	$\mathbf{W_{0.52}/D_{0.25}}$	$\mathbf{W_{0.53}/D_{0.25}}$		$W_{0.02}/D_{0.17}$	$W_{0.07}/D_{0.23}$	$\mathbf{W_{0.42}/D_{0.19}}$
MCTS - HE,K	$\mathbf{W_{1.00}/D_{0.00}}$	$\mathbf{W_{0.90}/D_{0.07}}$	$\mathbf{W_{0.88}/D_{0.05}}$	$\mathbf{W_{0.82}/D_{0.17}}$		$\mathbf{W_{0.58}/D_{0.12}}$	$\mathbf{W_{0.84}/D_{0.08}}$
MCTS - HE,R	$\mathbf{W_{0.97}/D_{0.02}}$	$\mathbf{W_{0.78}/D_{0.15}}$	$\mathbf{W_{0.78}/D_{0.13}}$	$\mathbf{W_{0.70}/D_{0.23}}$	$W_{0.30}/D_{0.12}$		$\mathbf{W_{0.71}/D_{0.13}}$

Figure A.5: Results for experiment *Expansion*, given 5000ms.

Appendix B

Experimental Results: Berlin - Heuristics

The following chapter contains the experimental results for the comparison of various playout policies in the domain of Berlin. Each table shows the win ratio (W) and draw ratio (D) per match up. If the row algorithm outperforms the column algorithm, defined as $W + \frac{D}{2} \geq 0.5$, then the entry is marked as bold. The experimental setup of each datapoint is **60** matches, equally divided over all combinations for starting positions. The matches are played on the *2 Player - Standard* map.

Row *vs* Column	**Genghis Khan**	**MCTS**	**MCTS HE, .25**	**MCTS HE, .75**	**MCTS HE, GenghisKhan**	**MCTS HE, Random**	**Random**	*Total*
Genghis Khan		**W1.00/D0.00**	**W1.00/D0.00**	**W0.68/D0.32**	**W0.25/D0.65**	**W1.00/D0.00**	**W1.00/D0.00**	**W0.82/D0.16**
MCTS	W0.00/D0.00		W0.02/D0.08	W0.02/D0.07	W0.05/D0.02	W0.05/D0.07	W0.37/D0.17	W0.08/D0.07
MCTS - HE, .25	W0.00/D0.00	**W0.90/D0.08**		W0.25/D0.22	**W0.35/D0.32**	**W0.28/D0.45**	**W0.83/D0.08**	**W0.44/D0.19**
MCTS - HE, .75	W0.00/D0.32	**W0.92/D0.07**	**W0.53/D0.22**		**W0.53/D0.27**	**W0.38/D0.40**	**W0.85/D0.10**	**W0.54/D0.23**
MCTS - HE, GenghisKhan	W0.10/D0.65	**W0.93/D0.02**	W0.33/D0.32	W0.20/D0.27		W0.23/D0.45	**W0.95/D0.02**	**W0.46/D0.29**
MCTS - HE, Random	W0.00/D0.00	**W0.88/D0.07**	W0.27/D0.45	W0.22/D0.40	**W0.32/D0.45**		**W0.88/D0.10**	**W0.43/D0.24**
Random	W0.00/D0.00	**W0.47/D0.17**	W0.08/D0.08	W0.05/D0.10	W0.03/D0.02	W0.02/D0.10		W0.11/D0.08

Figure B.1: Results for experiment *Heuristics*, given 250ms.

Row *vs* Column	**Genghis Khan**	**MCTS**	**MCTS HE, .25**	**MCTS HE, .75**	**MCTS HE, GenghisKhan**	**MCTS HE, Random**	**Random**	*Total*
Genghis Khan		**W1.00/D0.00**	**W0.83/D0.15**	**W0.53/D0.30**	**W0.23/D0.75**	**W0.88/D0.08**	**W1.00/D0.00**	**W0.75/D0.21**
MCTS	W0.00/D0.00		W0.07/D0.05	W0.02/D0.03	W0.00/D0.05	W0.00/D0.00	**W0.53/D0.15**	W0.10/D0.05
MCTS - HE, .25	W0.02/D0.15	**W0.88/D0.05**		W0.23/D0.32	W0.30/D0.35	W0.17/D0.62	**W0.93/D0.05**	**W0.42/D0.26**
MCTS - HE, .75	W0.17/D0.30	**W0.95/D0.03**	**W0.45/D0.32**		**W0.58/D0.27**	**W0.48/D0.33**	**W1.00/D0.00**	**W0.61/D0.21**
MCTS - HE, GenghisKhan	W0.02/D0.75	**W0.95/D0.05**	**W0.35/D0.35**	W0.15/D0.27		W0.32/D0.28	**W0.88/D0.03**	**W0.44/D0.29**
MCTS - HE, Random	W0.03/D0.08	**W1.00/D0.00**	**W0.22/D0.62**	W0.18/D0.33	**W0.40/D0.28**		**W0.93/D0.05**	**W0.46/D0.23**
Random	W0.00/D0.00	W0.32/D0.15	W0.02/D0.05	W0.00/D0.00	W0.08/D0.03	W0.02/D0.05		W0.07/D0.05

Figure B.2: Results for experiment *Heuristics*, given 500ms.

Row *vs* Column	**Genghis Khan**	**MCTS**	**MCTS HE, .25**	**MCTS HE, .75**	**MCTS HE, GenghisKhan**	**MCTS HE, Random**	**Random**	*Total*
Genghis Khan		**W1.00/D0.00**	**W0.73/D0.22**	**W0.40/D0.35**	**W0.28/D0.55**	**W0.80/D0.13**	**W1.00/D0.00**	**W0.70/D0.21**
MCTS	W0.00/D0.00		W0.00/D0.00	W0.00/D0.02	W0.00/D0.03	W0.00/D0.05	**W0.43/D0.17**	W0.07/D0.04
MCTS - HE, .25	W0.05/D0.22	**W1.00/D0.00**		W0.32/D0.35	**W0.57/D0.32**	W0.25/D0.42	**W1.00/D0.00**	**W0.53/D0.22**
MCTS - HE, .75	W0.25/D0.35	**W0.98/D0.02**	**W0.33/D0.35**		**W0.75/D0.17**	**W0.38/D0.45**	**W1.00/D0.00**	**W0.62/D0.22**
MCTS - HE, GenghisKhan	W0.17/D0.55	**W0.97/D0.03**	W0.12/D0.32	W0.08/D0.17		W0.28/D0.33	**W0.92/D0.08**	**W0.42/D0.25**
MCTS - HE, Random	W0.07/D0.13	**W0.95/D0.05**	**W0.33/D0.42**	W0.17/D0.45	**W0.38/D0.33**		**W0.95/D0.05**	**W0.47/D0.24**
Random	W0.00/D0.00	W0.40/D0.17	W0.00/D0.00	W0.00/D0.00	W0.00/D0.08	W0.00/D0.05		W0.07/D0.05

Figure B.3: Results for experiment *Heuristics*, given 1000ms.

Row *vs* Column	**Genghis Khan**	**MCTS**	**MCTS HE, .25**	**MCTS HE, .75**	**MCTS HE, GenghisKhan**	**MCTS HE, Random**	**Random**	*Total*
Genghis Khan		**W1.00/D0.00**	**W0.47/D0.38**	W0.20/D0.30	W0.12/D0.75	**W0.67/D0.27**	**W1.00/D0.00**	**W0.57/D0.28**
MCTS	W0.00/D0.00		W0.00/D0.00	W0.00/D0.02	W0.07/D0.05	W0.00/D0.03	**W0.53/D0.02**	W0.10/D0.02
MCTS - HE, .25	W0.15/D0.38	**W1.00/D0.00**		W0.18/D0.37	**W0.62/D0.28**	**W0.22/D0.60**	**W1.00/D0.00**	**W0.53/D0.27**
MCTS - HE, .75	**W0.50/D0.30**	**W0.98/D0.02**	**W0.45/D0.37**		**W0.77/D0.20**	**W0.45/D0.38**	**W1.00/D0.00**	**W0.69/D0.21**
MCTS - HE, GenghisKhan	**W0.13/D0.75**	**W0.88/D0.05**	W0.10/D0.28	W0.03/D0.20		W0.22/D0.38	**W0.95/D0.05**	**W0.39/D0.29**
MCTS - HE, Random	W0.07/D0.27	**W0.97/D0.03**	W0.18/D0.60	W0.17/D0.38	**W0.40/D0.38**		**W0.98/D0.00**	**W0.46/D0.28**
Random	W0.00/D0.00	W0.45/D0.02	W0.00/D0.00	W0.00/D0.00	W0.00/D0.05	W0.02/D0.00		W0.08/D0.01

Figure B.4: Results for experiment *Heuristics*, given 2500ms.

Row *vs* Column	**Genghis Khan**	**MCTS**	**MCTS HE, .25**	**MCTS HE, .75**	**MCTS HE, GenghisKhan**	**MCTS HE, Random**	**Random**	*Total*
Genghis Khan		**W1.00/D0.00**	**W0.38/D0.30**	W0.17/D0.13	W0.20/D0.53	**W0.42/D0.38**	**W1.00/D0.00**	**W0.53/D0.22**
MCTS	W0.00/D0.00		W0.00/D0.02	W0.02/D0.00	W0.00/D0.03	W0.00/D0.00	W0.42/D0.13	W0.07/D0.03
MCTS - HE, .25	W0.32/D0.30	**W0.98/D0.02**		W0.22/D0.32	**W0.77/D0.13**	**W0.33/D0.42**	**W1.00/D0.00**	**W0.60/D0.20**
MCTS - HE, .75	**W0.70/D0.13**	**W0.98/D0.00**	**W0.47/D0.32**		**W0.77/D0.20**	**W0.42/D0.32**	**W1.00/D0.00**	**W0.72/D0.16**
MCTS - HE, GenghisKhan	**W0.27/D0.53**	**W0.97/D0.03**	W0.10/D0.13	W0.03/D0.20		W0.15/D0.25	**W0.97/D0.03**	**W0.41/D0.20**
MCTS - HE, Random	W0.20/D0.38	**W1.00/D0.00**	W0.25/D0.42	W0.27/D0.32	**W0.60/D0.25**		**W1.00/D0.00**	**W0.55/D0.23**
Random	W0.00/D0.00	**W0.45/D0.13**	W0.00/D0.00	W0.00/D0.00	W0.00/D0.03	W0.00/D0.00		W0.08/D0.03

Figure B.5: Results for experiment *Heuristics*, given 5000ms.

Appendix C

Experimental Results: Berlin - Search Algorithms

The following chapter contains the experimental results for the comparison of various algorithms in the domain of Berlin. Each table shows the win ratio (W) and draw ratio (D) per match up. If the row algorithm outperforms the column algorithm, defined as $W + \frac{D}{2} \geq 0.5$, then the entry is marked as bold. The experimental setup of each datapoint is **100** matches, equally divided over all combinations for starting positions. The matches are played on the *2 Player - Standard* map.

Row *vs* Column	**Genghis Khan**	**LSI alt**	**LSI tl**	**MCTS**	**MCTS DE**	**MCTS HE,K**	**NMC**	**Random**	*Total*
Genghis Khan		**W0.96/D0.04**	**W1.00/D0.00**	**W1.00/D0.00**	**W1.00/D0.00**	**W0.63/D0.26**	**W0.89/D0.11**	**W1.00/D0.00**	**W0.92/D0.06**
LSI - alt	W0.00/D0.04		**W0.72/D0.19**	**W1.00/D0.00**	**W0.63/D0.18**	W0.16/D0.22	W0.28/D0.28	**W0.77/D0.12**	**W0.51/D0.15**
LSI - tl	W0.00/D0.00	W0.09/D0.19		**W1.00/D0.00**	W0.34/D0.16	W0.04/D0.12	W0.08/D0.05	**W0.53/D0.08**	W0.31/D0.08
MCTS	W0.00/D0.00	W0.00/D0.00	W0.00/D0.00		W0.00/D0.00	W0.00/D0.01	W0.00/D0.00	W0.00/D0.00	W0.00/D0.00
MCTS - DE	W0.00/D0.00	W0.19/D0.18	**W0.50/D0.16**	**W1.00/D0.00**		W0.06/D0.09	W0.07/D0.14	**W0.68/D0.10**	W0.36/D0.09
MCTS - HE,K	W0.11/D0.26	**W0.62/D0.22**	**W0.84/D0.12**	**W0.99/D0.01**	**W0.85/D0.09**		W0.36/D0.25	**W0.97/D0.03**	**W0.68/D0.14**
NMC	W0.00/D0.11	**W0.45/D0.28**	**W0.87/D0.05**	**W1.00/D0.00**	**W0.79/D0.14**	**W0.39/D0.25**		**W0.94/D0.03**	**W0.64/D0.12**
Random	W0.00/D0.00	W0.12/D0.12	W0.39/D0.08	**W1.00/D0.00**	W0.22/D0.10	W0.00/D0.03	W0.03/D0.03		W0.26/D0.05

Figure C.1: Results for experiment *Algorithms*, given 250ms.

Row *vs* Column	**Genghis Khan**	**LSI alt**	**LSI tl**	**MCTS**	**MCTS DE**	**MCTS HE,K**	**NMC**	**Random**	*Total*
Genghis Khan		**W0.96/D0.04**	**W1.00/D0.00**	**W1.00/D0.00**	**W0.99/D0.01**	**W0.38/D0.45**	**W0.95/D0.05**	**W1.00/D0.00**	**W0.90/D0.08**
LSI - alt	W0.00/D0.04		**W0.60/D0.16**	**W0.98/D0.01**	**W0.61/D0.20**	W0.16/D0.20	W0.24/D0.25	**W0.83/D0.05**	**W0.50/D0.13**
LSI - tl	W0.00/D0.00	W0.24/D0.16		**W1.00/D0.00**	**W0.45/D0.26**	W0.04/D0.15	W0.10/D0.19	**W0.59/D0.14**	W0.35/D0.13
MCTS	W0.00/D0.00	W0.01/D0.01	W0.00/D0.00		W0.00/D0.00	W0.00/D0.00	W0.00/D0.00	W0.00/D0.01	W0.00/D0.00
MCTS - DE	W0.00/D0.01	W0.18/D0.20	W0.29/D0.26	**W1.00/D0.00**		W0.03/D0.09	W0.01/D0.05	**W0.75/D0.08**	W0.32/D0.10
MCTS - HE,K	W0.17/D0.45	**W0.64/D0.20**	**W0.81/D0.15**	**W1.00/D0.00**	**W0.88/D0.09**		**W0.49/D0.20**	**W0.99/D0.01**	**W0.71/D0.16**
NMC	W0.00/D0.05	**W0.52/D0.25**	**W0.71/D0.19**	**W1.00/D0.00**	**W0.94/D0.05**	W0.31/D0.20		**W0.98/D0.02**	**W0.64/D0.11**
Random	W0.00/D0.00	W0.12/D0.05	W0.27/D0.14	**W0.99/D0.01**	W0.17/D0.08	W0.00/D0.01	W0.00/D0.02		W0.22/D0.04

Figure C.2: Results for experiment *Algorithms*, given 500ms.

Row *vs* Column	**Genghis Khan**	**LSI alt**	**LSI tl**	**MCTS**	**MCTS DE**	**MCTS HE,K**	**NMC**	**Random**	*Total*
Genghis Khan		**W0.92/D0.06**	**W0.97/D0.03**	**W1.00/D0.00**	**W0.97/D0.02**	W0.23/D0.38	**W0.91/D0.06**	**W1.00/D0.00**	**W0.86/D0.08**
LSI - alt	W0.01/D0.06		**W0.53/D0.21**	**W0.98/D0.02**	**W0.75/D0.14**	W0.09/D0.22	W0.24/D0.17	**W0.88/D0.06**	**W0.50/D0.13**
LSI - tl	W0.00/D0.03	W0.26/D0.21		**W1.00/D0.00**	W0.37/D0.26	W0.06/D0.19	W0.07/D0.14	**W0.73/D0.12**	W0.36/D0.14
MCTS	W0.00/D0.00	W0.00/D0.02	W0.00/D0.00		W0.00/D0.00	W0.00/D0.00	W0.00/D0.00	W0.00/D0.01	W0.00/D0.00
MCTS - DE	W0.01/D0.02	W0.11/D0.14	W0.37/D0.26	**W1.00/D0.00**		W0.01/D0.09	W0.08/D0.07	**W0.77/D0.09**	W0.34/D0.10
MCTS - HE,K	**W0.39/D0.38**	**W0.69/D0.22**	**W0.75/D0.19**	**W1.00/D0.00**	**W0.90/D0.09**		**W0.52/D0.28**	**W0.99/D0.01**	**W0.75/D0.17**
NMC	W0.03/D0.06	**W0.59/D0.17**	**W0.79/D0.14**	**W1.00/D0.00**	**W0.85/D0.07**	W0.20/D0.28		**W0.96/D0.03**	**W0.63/D0.11**
Random	W0.00/D0.00	W0.06/D0.06	W0.15/D0.12	**W0.99/D0.01**	W0.14/D0.09	W0.00/D0.01	W0.01/D0.03		W0.19/D0.05

Figure C.3: Results for experiment *Algorithms*, given 1000ms.

Row *vs* Column	**Genghis Khan**	**LSI alt**	**LSI tl**	**MCTS**	**MCTS DE**	**MCTS HE,K**	**NMC**	**Random**	*Total*
Genghis Khan		**W0.98/D0.01**	**W0.98/D0.02**	**W1.00/D0.00**	**W0.89/D0.11**	W0.16/D0.22	**W0.77/D0.10**	**W1.00/D0.00**	**W0.82/D0.07**
LSI - alt	W0.01/D0.01		**W0.51/D0.27**	**W0.96/D0.04**	**W0.56/D0.25**	W0.08/D0.20	W0.11/D0.19	**W0.89/D0.07**	**W0.45/D0.15**
LSI - tl	W0.00/D0.02	W0.22/D0.27		**W0.98/D0.01**	W0.39/D0.19	W0.06/D0.10	W0.05/D0.10	**W0.75/D0.10**	W0.35/D0.11
MCTS	W0.00/D0.00	W0.00/D0.04	W0.01/D0.01		W0.00/D0.00	W0.00/D0.01	W0.00/D0.01	W0.04/D0.04	W0.01/D0.02
MCTS - DE	W0.00/D0.11	W0.19/D0.25	**W0.42/D0.19**	**W1.00/D0.00**		W0.01/D0.09	W0.07/D0.11	**W0.84/D0.07**	W0.36/D0.12
MCTS - HE,K	**W0.62/D0.22**	**W0.72/D0.20**	**W0.84/D0.10**	**W0.99/D0.01**	**W0.90/D0.09**		**W0.59/D0.22**	**W1.00/D0.00**	**W0.81/D0.12**
NMC	W0.13/D0.10	**W0.70/D0.19**	**W0.85/D0.10**	**W0.99/D0.01**	**W0.82/D0.11**	W0.19/D0.22		**W0.98/D0.02**	**W0.67/D0.11**
Random	W0.00/D0.00	W0.04/D0.07	W0.15/D0.10	**W0.92/D0.04**	W0.09/D0.07	W0.00/D0.00	W0.00/D0.02		W0.17/D0.04

Figure C.4: Results for experiment *Algorithms*, given 2500ms.

Row *vs* Column	**Genghis Khan**	**LSI alt**	**LSI tl**	**MCTS**	**MCTS DE**	**MCTS HE,K**	**NMC**	**Random**	*Total*
Genghis Khan		**W0.92/D0.06**	**W1.00/D0.00**	**W0.99/D0.01**	**W0.88/D0.10**	W0.12/D0.15	**W0.64/D0.20**	**W1.00/D0.00**	**W0.79/D0.07**
LSI - alt	W0.02/D0.06		**W0.49/D0.19**	**W0.96/D0.02**	**W0.57/D0.24**	W0.05/D0.08	W0.08/D0.14	**W0.94/D0.05**	**W0.45/D0.11**
LSI - tl	W0.00/D0.00	W0.32/D0.19		**W0.95/D0.02**	W0.34/D0.31	W0.04/D0.04	W0.04/D0.11	**W0.76/D0.13**	W0.35/D0.11
MCTS	W0.00/D0.01	W0.02/D0.02	W0.03/D0.02		W0.00/D0.00	W0.01/D0.00	W0.00/D0.01	W0.05/D0.04	W0.02/D0.01
MCTS - DE	W0.02/D0.10	W0.19/D0.24	**W0.35/D0.31**	**W1.00/D0.00**		W0.00/D0.03	W0.03/D0.10	**W0.79/D0.05**	W0.34/D0.12
MCTS - HE,K	**W0.73/D0.15**	**W0.87/D0.08**	**W0.92/D0.04**	**W0.99/D0.00**	**W0.97/D0.03**		**W0.61/D0.23**	**W1.00/D0.00**	**W0.87/D0.08**
NMC	W0.16/D0.20	**W0.78/D0.14**	**W0.85/D0.11**	**W0.99/D0.01**	**W0.87/D0.10**	W0.16/D0.23		**W0.98/D0.02**	**W0.68/D0.12**
Random	W0.00/D0.00	W0.01/D0.05	W0.11/D0.13	**W0.91/D0.04**	W0.16/D0.05	W0.00/D0.00	W0.00/D0.02		W0.17/D0.04

Figure C.5: Results for experiment *Algorithms*, given 5000ms.

Appendix D

Experimental Results: Berlin - Partial Move Completion

The following chapter contains the experimental results for the comparison of various policies for partial move completion in the domain of Berlin. Each table shows the win ratio (W) and draw ratio (D) per match up. If the row algorithm outperforms the column algorithm, defined as $W + \frac{D}{2} \geq 0.5$, then the entry is marked as bold. The experimental setup of each datapoint is **100** matches, equally divided over all combinations of starting positions. The matches are played on the *2 Player - Standard* map.

Row *vs* Column	MCTS DE	MCTS DE Heuristic	MCTS DE RAVE	MCTS HE	MCTS HE Heuristic	MCTS HE RAVE	MCTS HE,R	MCTS HE,R Heuristic	MCTS HE,R RAVE	*Total*
MCTS - DE		$W_{0.00}/D_{0.00}$	$W_{0.08}/D_{0.15}$	**$W_{0.62}/D_{0.12}$**	$W_{0.08}/D_{0.12}$	**$W_{0.64}/D_{0.12}$**	$W_{0.19}/D_{0.18}$	$W_{0.01}/D_{0.03}$	$W_{0.02}/D_{0.11}$	$W_{0.20}/D_{0.10}$
MCTS - DE - Heuristic	**$W_{1.00}/D_{0.00}$**		**$W_{0.90}/D_{0.07}$**	**$W_{0.99}/D_{0.01}$**	**$W_{0.95}/D_{0.04}$**	**$W_{1.00}/D_{0.00}$**	**$W_{0.94}/D_{0.05}$**	**$W_{0.49}/D_{0.18}$**	**$W_{0.80}/D_{0.12}$**	**$W_{0.88}/D_{0.06}$**
MCTS - DE - RAVE	**$W_{0.77}/D_{0.15}$**	$W_{0.03}/D_{0.07}$		**$W_{0.82}/D_{0.12}$**	$W_{0.33}/D_{0.28}$	**$W_{0.86}/D_{0.08}$**	**$W_{0.51}/D_{0.24}$**	$W_{0.11}/D_{0.14}$	$W_{0.38}/D_{0.22}$	**$W_{0.48}/D_{0.16}$**
MCTS - HE	$W_{0.26}/D_{0.12}$	$W_{0.00}/D_{0.01}$	$W_{0.06}/D_{0.12}$		$W_{0.08}/D_{0.07}$	**$W_{0.52}/D_{0.10}$**	$W_{0.13}/D_{0.20}$	$W_{0.00}/D_{0.03}$	$W_{0.05}/D_{0.03}$	$W_{0.14}/D_{0.09}$
MCTS - HE - Heuristic	**$W_{0.80}/D_{0.12}$**	$W_{0.01}/D_{0.04}$	**$W_{0.39}/D_{0.28}$**	**$W_{0.85}/D_{0.07}$**		**$W_{0.90}/D_{0.05}$**	**$W_{0.60}/D_{0.16}$**	$W_{0.04}/D_{0.09}$	$W_{0.28}/D_{0.31}$	**$W_{0.48}/D_{0.14}$**
MCTS - HE - RAVE	$W_{0.24}/D_{0.12}$	$W_{0.00}/D_{0.00}$	$W_{0.06}/D_{0.08}$	$W_{0.38}/D_{0.10}$	$W_{0.05}/D_{0.05}$		$W_{0.09}/D_{0.14}$	$W_{0.01}/D_{0.02}$	$W_{0.03}/D_{0.07}$	$W_{0.11}/D_{0.07}$
MCTS - HE,R	**$W_{0.63}/D_{0.18}$**	$W_{0.01}/D_{0.05}$	$W_{0.25}/D_{0.24}$	**$W_{0.67}/D_{0.20}$**	$W_{0.24}/D_{0.16}$	**$W_{0.77}/D_{0.14}$**		$W_{0.05}/D_{0.14}$	$W_{0.23}/D_{0.15}$	$W_{0.36}/D_{0.16}$
MCTS - HE,R - Heuristic	**$W_{0.96}/D_{0.03}$**	$W_{0.33}/D_{0.18}$	**$W_{0.75}/D_{0.14}$**	**$W_{0.97}/D_{0.03}$**	**$W_{0.87}/D_{0.09}$**	**$W_{0.97}/D_{0.02}$**	**$W_{0.81}/D_{0.14}$**		**$W_{0.74}/D_{0.20}$**	**$W_{0.80}/D_{0.10}$**
MCTS - HE,R - RAVE	**$W_{0.87}/D_{0.11}$**	$W_{0.08}/D_{0.12}$	**$W_{0.40}/D_{0.22}$**	**$W_{0.92}/D_{0.03}$**	**$W_{0.41}/D_{0.31}$**	**$W_{0.90}/D_{0.07}$**	**$W_{0.62}/D_{0.15}$**	$W_{0.06}/D_{0.20}$		**$W_{0.53}/D_{0.15}$**

Figure D.1: Results for experiment *Partial move*, given 250ms.

Row *vs* Column	MCTS DE	MCTS DE Heuristic	MCTS DE RAVE	MCTS HE	MCTS HE Heuristic	MCTS HE RAVE	MCTS HE,R	MCTS HE,R Heuristic	MCTS HE,R RAVE	*Total*
MCTS - DE		$W_{0.00}/D_{0.01}$	$W_{0.06}/D_{0.14}$	**$W_{0.44}/D_{0.16}$**	$W_{0.07}/D_{0.15}$	**$W_{0.47}/D_{0.15}$**	$W_{0.08}/D_{0.21}$	$W_{0.03}/D_{0.03}$	$W_{0.03}/D_{0.07}$	$W_{0.15}/D_{0.12}$
MCTS - DE - Heuristic	**$W_{0.99}/D_{0.01}$**		**$W_{0.82}/D_{0.15}$**	**$W_{0.99}/D_{0.01}$**	**$W_{0.89}/D_{0.07}$**	**$W_{0.98}/D_{0.02}$**	**$W_{0.91}/D_{0.07}$**	**$W_{0.50}/D_{0.20}$**	**$W_{0.70}/D_{0.22}$**	**$W_{0.85}/D_{0.09}$**
MCTS - DE - RAVE	**$W_{0.80}/D_{0.14}$**	$W_{0.03}/D_{0.15}$		**$W_{0.84}/D_{0.11}$**	**$W_{0.39}/D_{0.27}$**	**$W_{0.83}/D_{0.11}$**	**$W_{0.56}/D_{0.26}$**	$W_{0.12}/D_{0.24}$	$W_{0.25}/D_{0.24}$	**$W_{0.48}/D_{0.19}$**
MCTS - HE	$W_{0.40}/D_{0.16}$	$W_{0.00}/D_{0.01}$	$W_{0.05}/D_{0.11}$		$W_{0.06}/D_{0.15}$	**$W_{0.49}/D_{0.15}$**	$W_{0.13}/D_{0.18}$	$W_{0.02}/D_{0.04}$	$W_{0.01}/D_{0.08}$	$W_{0.14}/D_{0.11}$
MCTS - HE - Heuristic	**$W_{0.78}/D_{0.15}$**	$W_{0.04}/D_{0.07}$	$W_{0.34}/D_{0.27}$	**$W_{0.79}/D_{0.15}$**		**$W_{0.79}/D_{0.15}$**	**$W_{0.48}/D_{0.29}$**	$W_{0.06}/D_{0.18}$	$W_{0.24}/D_{0.26}$	**$W_{0.44}/D_{0.19}$**
MCTS - HE - RAVE	$W_{0.38}/D_{0.15}$	$W_{0.00}/D_{0.02}$	$W_{0.06}/D_{0.11}$	$W_{0.36}/D_{0.15}$	$W_{0.06}/D_{0.15}$		$W_{0.08}/D_{0.07}$	$W_{0.03}/D_{0.06}$	$W_{0.00}/D_{0.07}$	$W_{0.12}/D_{0.10}$
MCTS - HE,R	**$W_{0.71}/D_{0.21}$**	$W_{0.02}/D_{0.07}$	$W_{0.18}/D_{0.26}$	**$W_{0.69}/D_{0.18}$**	$W_{0.23}/D_{0.29}$	**$W_{0.85}/D_{0.07}$**		$W_{0.03}/D_{0.14}$	$W_{0.19}/D_{0.17}$	$W_{0.36}/D_{0.17}$
MCTS - HE,R - Heuristic	**$W_{0.94}/D_{0.03}$**	$W_{0.30}/D_{0.20}$	**$W_{0.64}/D_{0.24}$**	**$W_{0.94}/D_{0.04}$**	**$W_{0.76}/D_{0.18}$**	**$W_{0.91}/D_{0.06}$**	**$W_{0.83}/D_{0.14}$**		**$W_{0.59}/D_{0.28}$**	**$W_{0.74}/D_{0.15}$**
MCTS - HE,R - RAVE	**$W_{0.90}/D_{0.07}$**	$W_{0.08}/D_{0.22}$	**$W_{0.51}/D_{0.24}$**	**$W_{0.91}/D_{0.08}$**	**$W_{0.50}/D_{0.26}$**	**$W_{0.93}/D_{0.07}$**	**$W_{0.64}/D_{0.17}$**	$W_{0.13}/D_{0.28}$		**$W_{0.57}/D_{0.17}$**

Figure D.2: Results for experiment *Partial move*, given 500ms.

Row *vs* Column	MCTS DE	MCTS DE Heuristic	MCTS DE RAVE	MCTS HE	MCTS HE Heuristic	MCTS HE RAVE	MCTS HE,R	MCTS HE,R Heuristic	MCTS HE,R RAVE	*Total*
MCTS - DE		$W_{0.00}/D_{0.06}$	$W_{0.04}/D_{0.12}$	$W_{0.40}/D_{0.17}$	$W_{0.03}/D_{0.09}$	**$W_{0.45}/D_{0.16}$**	$W_{0.10}/D_{0.22}$	$W_{0.01}/D_{0.05}$	$W_{0.01}/D_{0.11}$	$W_{0.13}/D_{0.12}$
MCTS - DE - Heuristic	**$W_{0.94}/D_{0.06}$**		**$W_{0.78}/D_{0.15}$**	**$W_{0.95}/D_{0.05}$**	**$W_{0.78}/D_{0.13}$**	**$W_{0.94}/D_{0.05}$**	**$W_{0.86}/D_{0.10}$**	**$W_{0.39}/D_{0.26}$**	**$W_{0.69}/D_{0.15}$**	**$W_{0.79}/D_{0.12}$**
MCTS - DE - RAVE	**$W_{0.84}/D_{0.12}$**	$W_{0.07}/D_{0.15}$		**$W_{0.83}/D_{0.12}$**	**$W_{0.39}/D_{0.27}$**	**$W_{0.88}/D_{0.06}$**	**$W_{0.54}/D_{0.25}$**	$W_{0.14}/D_{0.23}$	$W_{0.33}/D_{0.31}$	**$W_{0.50}/D_{0.19}$**
MCTS - HE	**$W_{0.43}/D_{0.17}$**	$W_{0.00}/D_{0.05}$	$W_{0.05}/D_{0.12}$		$W_{0.10}/D_{0.15}$	**$W_{0.51}/D_{0.15}$**	$W_{0.09}/D_{0.17}$	$W_{0.00}/D_{0.04}$	$W_{0.00}/D_{0.10}$	$W_{0.15}/D_{0.12}$
MCTS - HE - Heuristic	**$W_{0.88}/D_{0.09}$**	$W_{0.09}/D_{0.13}$	$W_{0.34}/D_{0.27}$	**$W_{0.75}/D_{0.15}$**		**$W_{0.82}/D_{0.13}$**	**$W_{0.45}/D_{0.30}$**	$W_{0.16}/D_{0.16}$	$W_{0.13}/D_{0.34}$	**$W_{0.45}/D_{0.20}$**
MCTS - HE - RAVE	$W_{0.39}/D_{0.16}$	$W_{0.01}/D_{0.05}$	$W_{0.06}/D_{0.06}$	$W_{0.34}/D_{0.15}$	$W_{0.05}/D_{0.13}$		$W_{0.10}/D_{0.06}$	$W_{0.00}/D_{0.05}$	$W_{0.03}/D_{0.08}$	$W_{0.12}/D_{0.09}$
MCTS - HE,R	**$W_{0.68}/D_{0.22}$**	$W_{0.04}/D_{0.10}$	$W_{0.21}/D_{0.25}$	**$W_{0.74}/D_{0.17}$**	$W_{0.25}/D_{0.30}$	**$W_{0.84}/D_{0.06}$**		$W_{0.03}/D_{0.15}$	$W_{0.13}/D_{0.23}$	$W_{0.37}/D_{0.19}$
MCTS - HE,R - Heuristic	**$W_{0.94}/D_{0.05}$**	$W_{0.35}/D_{0.26}$	**$W_{0.63}/D_{0.23}$**	**$W_{0.96}/D_{0.04}$**	**$W_{0.68}/D_{0.16}$**	**$W_{0.95}/D_{0.05}$**	**$W_{0.82}/D_{0.15}$**		**$W_{0.46}/D_{0.25}$**	**$W_{0.72}/D_{0.15}$**
MCTS - HE,R - RAVE	**$W_{0.88}/D_{0.11}$**	$W_{0.16}/D_{0.15}$	**$W_{0.36}/D_{0.31}$**	**$W_{0.90}/D_{0.10}$**	**$W_{0.53}/D_{0.34}$**	**$W_{0.89}/D_{0.08}$**	**$W_{0.64}/D_{0.23}$**	$W_{0.29}/D_{0.25}$		**$W_{0.58}/D_{0.20}$**

Figure D.3: Results for experiment *Partial move*, given 1000ms.

Row *vs* Column	**MCTS DE**	**MCTS DE Heuristic**	**MCTS DE RAVE**	**MCTS HE**	**MCTS HE Heuristic**	**MCTS HE RAVE**	**MCTS HE,R**	**MCTS HE,R Heuristic**	**MCTS HE,R RAVE**	*Total*
MCTS - DE		$W0.02/D0.06$	$W0.02/D0.16$	$W0.26/D0.24$	$W0.01/D0.10$	$W0.32/D0.18$	$W0.06/D0.14$	$W0.01/D0.06$	$W0.01/D0.06$	$W0.09/D0.13$
MCTS - DE - Heuristic	**W0.92/D0.06**		**W0.56/D0.24**	**W0.81/D0.15**	**W0.59/D0.20**	**W0.96/D0.03**	**W0.71/D0.18**	$W0.35/D0.22$	**W0.42/D0.18**	**W0.67/D0.16**
MCTS - DE - RAVE	**W0.82/D0.16**	$W0.20/D0.24$		**W0.81/D0.15**	**W0.47/D0.25**	**W0.87/D0.12**	**W0.47/D0.32**	$W0.16/D0.21$	$W0.23/D0.30$	**W0.50/D0.22**
MCTS - HE	**W0.50/D0.24**	$W0.04/D0.15$	$W0.04/D0.15$		$W0.09/D0.06$	**W0.43/D0.24**	$W0.17/D0.20$	$W0.03/D0.08$	$W0.03/D0.12$	$W0.17/D0.16$
MCTS - HE - Heuristic	**W0.89/D0.10**	$W0.21/D0.20$	$W0.28/D0.25$	**W0.85/D0.06**		**W0.81/D0.13**	**W0.44/D0.32**	$W0.16/D0.19$	$W0.12/D0.20$	**W0.47/D0.18**
MCTS - HE - RAVE	**W0.50/D0.18**	$W0.01/D0.03$	$W0.01/D0.12$	$W0.33/D0.24$	$W0.06/D0.13$		$W0.07/D0.16$	$W0.04/D0.05$	$W0.03/D0.10$	$W0.13/D0.13$
MCTS - HE,R	**W0.80/D0.14**	$W0.11/D0.18$	$W0.21/D0.32$	**W0.63/D0.20**	$W0.24/D0.32$	**W0.77/D0.16**		$W0.05/D0.21$	$W0.21/D0.30$	$W0.38/D0.23$
MCTS - HE,R - Heuristic	**W0.93/D0.06**	**W0.43/D0.22**	**W0.63/D0.21**	**W0.89/D0.08**	**W0.65/D0.19**	**W0.91/D0.05**	**W0.74/D0.21**		**W0.45/D0.32**	**W0.70/D0.17**
MCTS - HE,R - RAVE	**W0.93/D0.06**	$W0.40/D0.18$	**W0.47/D0.30**	**W0.85/D0.12**	**W0.68/D0.20**	**W0.87/D0.10**	**W0.49/D0.30**	$W0.23/D0.32$		**W0.62/D0.20**

Figure D.4: Results for experiment *Partial move*, given 2500ms.

Row *vs* Column	**MCTS DE**	**MCTS DE Heuristic**	**MCTS DE RAVE**	**MCTS HE**	**MCTS HE Heuristic**	**MCTS HE RAVE**	**MCTS HE,R**	**MCTS HE,R Heuristic**	**MCTS HE,R RAVE**	*Total*
MCTS - DE		$W0.01/D0.03$	$W0.01/D0.12$	$W0.16/D0.32$	$W0.02/D0.07$	$W0.17/D0.29$	$W0.02/D0.08$	$W0.01/D0.05$	$W0.01/D0.02$	$W0.05/D0.12$
MCTS - DE - Heuristic	**W0.96/D0.03**		**W0.51/D0.22**	**W0.86/D0.12**	**W0.50/D0.21**	**W0.87/D0.10**	**W0.55/D0.22**	$W0.19/D0.19$	$W0.22/D0.18$	**W0.58/D0.16**
MCTS - DE - RAVE	**W0.87/D0.12**	$W0.27/D0.22$		**W0.78/D0.19**	**W0.40/D0.28**	**W0.80/D0.14**	**W0.41/D0.35**	$W0.16/D0.18$	$W0.19/D0.34$	**W0.49/D0.23**
MCTS - HE	**W0.52/D0.32**	$W0.02/D0.12$	$W0.03/D0.19$		$W0.09/D0.17$	$W0.38/D0.22$	$W0.11/D0.14$	$W0.02/D0.09$	$W0.02/D0.13$	$W0.15/D0.17$
MCTS - HE - Heuristic	**W0.91/D0.07**	$W0.29/D0.21$	$W0.32/D0.28$	**W0.74/D0.17**		**W0.68/D0.23**	**W0.44/D0.22**	$W0.14/D0.20$	$W0.13/D0.13$	**W0.46/D0.19**
MCTS - HE - RAVE	**W0.54/D0.29**	$W0.03/D0.10$	$W0.06/D0.14$	**W0.40/D0.22**	$W0.09/D0.23$		$W0.09/D0.13$	$W0.00/D0.09$	$W0.06/D0.08$	$W0.16/D0.16$
MCTS - HE,R	**W0.90/D0.08**	$W0.23/D0.22$	$W0.24/D0.35$	**W0.75/D0.14**	$W0.34/D0.22$	**W0.78/D0.13**		$W0.18/D0.16$	$W0.09/D0.21$	**W0.44/D0.19**
MCTS - HE,R - Heuristic	**W0.94/D0.05**	**W0.62/D0.19**	**W0.66/D0.18**	**W0.89/D0.09**	**W0.66/D0.20**	**W0.91/D0.09**	**W0.66/D0.16**		**W0.40/D0.28**	**W0.72/D0.16**
MCTS - HE,R - RAVE	**W0.97/D0.02**	**W0.60/D0.18**	**W0.47/D0.34**	**W0.85/D0.13**	**W0.74/D0.13**	**W0.86/D0.08**	**W0.70/D0.21**	$W0.32/D0.28$		**W0.69/D0.17**

Figure D.5: Results for experiment *Partial move*, given 5000ms.