

RUBENS DE OLIVEIRA MORAES FILHO

**ASYMMETRIC ACTION ABSTRACTIONS
FOR REAL-TIME STRATEGY GAMES**

Dissertação apresentada a Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2019

RUBENS DE OLIVEIRA MORAES FILHO

**ASYMMETRIC ACTION ABSTRACTIONS
FOR REAL-TIME STRATEGY GAMES**

Dissertation presented to the Graduate
Program in Computer Science of the Fed-
eral University of Viçosa in partial fulfill-
ment of the requirements for the degree of
Master in Computer Science

VIÇOSA
MINAS GERAIS - BRASIL
February 2019

**Ficha catalográfica preparada pela Biblioteca Central da Universidade
Federal de Viçosa - Câmpus Viçosa**

T

M827a Moraes Filho, Rubens de Oliveira, 1989-
2019 Asymmetric action abstractions for real-time strategy games
/ Rubens de Oliveira Moraes Filho. – Viçosa, MG, 2019.
xi, 77 f. : il. (algumas color.) ; 29 cm.

Texto em inglês.

Inclui apêndices.

Orientador: Levi Henrique Santana de Lélis.

Dissertação (mestrado) - Universidade Federal de Viçosa.

Referências bibliográficas: f. 48-51.

1. Programação (Computadores). 2. Jogos - Programação.
3. Arquitetura de software. 4. Abstração. 5. Algoritmos.
I. Universidade Federal de Viçosa. Departamento de Informática.
Programa de Pós-Graduação em Ciência da Computação.
II. Título.

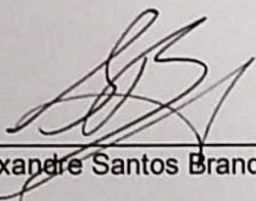
CDD 22. ed. 005.1

RUBENS DE OLIVEIRA MORAES FILHO

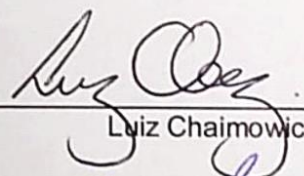
**ASYMMETRIC ACTION ABSTRACTIONS FOR REAL-TIME
STRATEGY GAMES**

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

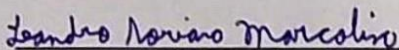
APROVADA: 08 de fevereiro de 2019.



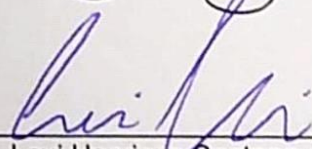
Alexandre Santos Brandão



Luiz Chaimowicz



Leandro Soriano Marcolino



Levi Henrique Santana de Lelis
(Orientador)

I would like to dedicate this dissertation to Roseana Maria Massaroni Moraes, mother and life example, who, for so many years, was my biggest motivator.

Acknowledgments

I would first like to express my special thanks of gratitude to my advisor Dr. Levi Henrique Santana de Lelis, who gave me the golden opportunity to do my research in Artificial Intelligence, at Universidade Federal de Viçosa, and worked with him and Dr. Mário A. Nascimento at University of Alberta, in one of the most incredible experience in my life. He consistently appoints me to the correct path and never was tired to teach me something or discuss an idea.

I would like to thank Bruna Manali Martins for shares with me your days.

I would like to thank Jeronimo Penha, Paulo Cândido, Betinho Corrêa, Jean Xavier and Julian Mariño for accepting nothing less than excellence from me.

Last but not least, I would like to thank my family: my parents and my sister, Raira Massaroni Moraes, for supporting me spiritually throughout during my master degree and my life in general.

Contents

List of Figures	vi
List of Tables	vii
Abstract	xi
1 Introduction	1
2 Related Work	5
2.1 Search-Based Approaches for RTS Games	5
2.2 Learning-Based Approaches for RTS Games	6
2.3 Action Abstractions in Extensive-Form Games	7
3 Background	8
3.1 Search Algorithms for Un-Abstracted Trees	10
3.1.1 Alpha-Beta Considering Durations (ABCD)	10
3.1.2 Naïve Monte Carlo Tree Search (NaïveMCTS)	12
3.2 Search Algorithms for Uniformly-Abstracted Trees	15
3.2.1 Uniform Action Abstractions	15
3.2.2 Portfolio Greedy Search	16
3.2.3 Stratified Strategy Selection	17
4 Asymmetric Action Abstractions	20
5 Searching in Asymmetrically-Abstracted Game Trees	23
5.1 Greedy and Stratified Alpha-Beta Searches (GAB and SAB)	24
5.1.1 Searching in Within-States	26
5.1.2 Baselines for GAB and SAB: $GAB_{\mathcal{P}}$, $SAB_{\mathcal{P}}$, GAS, and SAS	27
5.2 Asymmetrically Action-Abstracted NaïveMCTS (A3N)	28
5.2.1 Baselines for A3N: A1N and A2N	29

6	Empirical Evaluation Methodology and Settings Definition	31
6.1	μ RTS	31
6.1.1	Empirical Setting for μ RTS	33
6.2	Evaluating Within-State Search	35
6.2.1	Within-State Version of Baselines	37
6.3	Evaluating Strategies and Number of Unrestricted Units	37
6.4	Comparison with Baselines	41
6.5	Comparison with State-of-the-Art Algorithms	42
7	Conclusions	46
	Bibliography	48
	Appendix A Proofs	52
	Appendix B Evaluating Strategies and Number of Unrestricted Units for Each Map to GAB	54
	Appendix C Evaluating Strategies and Number of Unrestricted Units for Each Map to SAB	60
	Appendix D Evaluating Strategies and Number of Unrestricted Units for Each Map to A3N	66
	Appendix E Detailed Final Experiment	72

List of Figures

6.1	A μ RTS state of a game in a 8×8 map.	32
6.2	The within-state frequency of use by GAB and SAB.	37

List of Tables

6.1	Maps used in our experiments. The names are as they appear in the μ RTS codebase. We also show the size of the maps and the maximum number of game cycles for matches played in each map.	34
6.2	Winning rate of GAB w , SAB w , and A3N w against their baselines.	35
6.3	Winning rate of variants of GAB against PGS in 100 matches played in 10 maps, 10 matches for each map. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	39
6.4	Winning rate of the row player against the column player. Comparison of GAB and SAB with their baselines.	42
6.5	Winning rate of the row player against the column player. Comparison of A3N with its baselines.	42
6.6	Comparison of GAB, SAB, and A3N with current state-of-the-art search-based methods.	43
6.7	Winning rate of the row player against the column player, divided into small maps (8×8 and 16×6) and large maps (24×24 , 32×32 and 64×64).	44
B.1	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	54
B.2	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	55
B.3	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	55
B.4	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	56

B.5	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	56
B.6	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	57
B.7	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	57
B.8	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	58
B.9	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	58
B.10	Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	59
C.1	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	60
C.2	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	61
C.3	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	61
C.4	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	62
C.5	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	62
C.6	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	63

C.7	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	63
C.8	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	64
C.9	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	64
C.10	Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	65
D.1	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	66
D.2	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	67
D.3	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	67
D.4	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	68
D.5	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	68
D.6	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	69
D.7	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	69
D.8	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	70

D.9	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	70
D.10	Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).	71
E.1	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	72
E.2	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	73
E.3	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	73
E.4	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	74
E.5	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	74
E.6	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	75
E.7	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	75
E.8	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	76
E.9	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	76
E.10	Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.	77

Abstract

MORAES FILHO, Rubens de Oliveira, M.Sc., Universidade Federal de Viçosa, February, 2019. **Asymmetric Action Abstractions for Real-Time Strategy Games**. Adviser: Levi Henrique Santana de Lelis.

Action abstractions restrict the number of legal actions available for real-time planning in zero-sum extensive-form games, thus allowing algorithms to focus their search on a set of promising actions. Optimal strategies derived from un-abstracted game trees are guaranteed to be no worse than optimal strategies derived from action-abstracted game trees. In practice, however, due to real-time constraints and the game tree size, one is only able to derive good strategies in un-abstracted trees in small-scale games. In this paper, we introduce an action abstraction scheme we call asymmetric abstraction. Asymmetric abstractions can retain the un-abstracted trees' theoretical advantage over regularly abstracted trees while still allowing search algorithms to derive effective strategies, even in large-scale games. Further, asymmetric abstractions allow search algorithms to “pay more attention” in some aspects of the game by unevenly dividing the algorithm's search effort amongst different aspects of the game. In this paper we also introduce four algorithms that search in asymmetrically-abstracted game trees to evaluate the effectiveness of our abstraction schemes. Two of those are greedy-search approaches and they are called Greedy Alpha-Beta Search and Stratified Alpha-Beta Search. The other two are versions of an algorithm we call Asymmetrically Action-Abstracted NaïveMCTS (A2N and A3N) and combine Monte Carlo Tree Search (MCTS) algorithms, *naïve sampling* strategy for select actions' samples, and asymmetric abstraction scheme. In addition to the search algorithms, we also introduce several strategies for generating asymmetric abstractions. An extensive set of experiments in a real-time strategy game developed for research purposes shows that search algorithms using asymmetric abstractions are able to outperform all other search algorithms tested.

Chapter 1

Introduction

In real-time strategy (RTS) games the player controls dozens of units to collect resources, build structures, and battle the opponent. RTS games are excellent testbeds for Artificial Intelligence methods because they offer fast-paced environments, where players act simultaneously, and the number of legal actions grows exponentially with the number of units the player controls. Also, the time allowed for planning is on the order of milliseconds. In this dissertation we assume two-player deterministic games in which all units are visible to both players (i.e., the opponent’s units are not hidden as it happens in some RTS games).

A successful family of algorithms for planning in real time in RTS games uses what we call action abstractions to reduce the number of legal actions available to the player. In RTS games, player actions are represented as a vector of unit-actions, where each entry in the vector represents an action for a unit controlled by the player. Action abstractions reduce the number of legal actions a player can perform by reducing the number of legal actions each unit can perform. We use the word “action” if it is clear that we are referring to a player’s or to a unit’s action; we write “player action” or “unit-action” otherwise.

Churchill and Buro (2013) introduced a method for building action abstractions through what they called scripts. A script $\bar{\sigma}$ is a function mapping a game state s and a unit u to an action m for u . A set of scripts \mathcal{P} induces an action abstraction by restricting the set of legal actions of all units to actions returned by the scripts in \mathcal{P} . We call an action abstraction generated with Churchill and Buro’s scheme a uniform abstraction.

In theory, as we show in this dissertation, for zero-sum extensive-form games, players searching in un-abstracted trees are guaranteed to derive optimal strategies that are no worse than the optimal strategies derived from action-abstracted trees

for a given opponent. This is because the former has access to actions that are not available in action-abstracted spaces. Despite its theoretical disadvantage, in practice uniform abstractions can be successful, specially in large-scale combats [1]. This happens because RTS games can have very large game trees, and the problem’s real-time constraints often allow search algorithms to explore only a small fraction of all legal actions before deciding on which action to perform next—uniform abstractions allow algorithms to focus their search on actions deemed as promising by the set of scripts \mathcal{P} .

In this dissertation we introduce an action abstraction scheme we call asymmetric action abstractions (or asymmetric abstractions for short). In contrast with uniform abstractions that restrict the number of actions of all units, asymmetric abstractions restrict the number of actions of only a subset of units. We show that asymmetric abstractions can retain the un-abstracted trees’ theoretical advantage over uniformly abstracted ones while still allowing algorithms to derive effective strategies in practice, even in large-scale games. Another advantage of asymmetric abstractions is that they allow the search effort to be distributed unevenly amongst the units. This is important because some units might benefit more from finer plans (i.e., plans computed while accounting for a larger set of actions) than others (e.g., it could be advantageous to provide finer control to units with low hit points).

Another contribution of this dissertation is the introduction of four general-purpose algorithms that search in asymmetrically-abstracted trees: Greedy Alpha-Beta Search (GAB), Stratified Alpha-Beta Search (SAB), Asymmetrically Action-Abstracted NaïveMCTS (A2N and A3N). GAB and SAB are based on Minimax with Alpha-Beta pruning, Portfolio Greedy Search (PGS) [1], and Stratified Strategy Selection (SSS) [2]. PGS and SSS are algorithms designed for searching in uniformly-abstracted game trees. The other two algorithms, A2N and A3N, are based on NaïveMCTS, a search algorithm that uses combinatorial multi-armed bandits (CMAB) [3, 4] to search in un-abstracted trees. In addition to the two variants of NaïveMCTS that search in asymmetrically-abstracted trees, we also introduce a NaïveMCTS baseline that, similarly to PGS and SSS, searches in uniformly-abstracted trees; we call this baseline A1N.

We evaluate our algorithms in μ RTS [3], an RTS game developed for research purposes. μ RTS is a great testbed for our research because it offers an efficient forward model of the game, which is required by search-based approaches. Moreover, the game is much simpler than commercial video games, which allows us to evaluate different algorithms without the technical difficulties typical of commercial video

games. Finally, μ RTS codebase¹ contains most of the current state-of-the-art search-based methods, including the systems used in the latest μ RTS competition [5], thus facilitating our empirical evaluation. An extensive set of experiments show that our algorithms that search in asymmetrically-abstracted trees are able to outperform their counterparts that search un-abstracted and uniformly-abstracted trees.

Although we present our abstraction schemes and search algorithms in the context of RTS games, we believe our ideas and algorithms to also be applicable in other scenarios. For example, Tavares et al. (2018) use the concept of uniform abstractions with reinforcement learning (RL) algorithms; asymmetric action abstractions could possibly be used with RL algorithms. Asymmetric abstractions might also be directly applied to other application domains. For example, a robotic system that controls several actuators simultaneously while trying to accomplish a task can benefit from asymmetric abstractions. This is because some actuators might require a finer control than the others. To illustrate, the actuators controlling the arms of a robot planning a sequence of actions to open a door might need a “finer plan” than the actuators controlling the wheels of the robot, given that the robot is already in front of the door to be opened. Asymmetric action abstractions offer an approach that allows the planning system to focus on the arms of the robot rather than on its wheels.

This dissertation extends two conference publications [6, 7]. Moraes and Lelis (2018) introduced asymmetric abstractions and the versions of PGS and SSS for searching in such trees, GAB and SAB, while Moraes et al. (2018a) introduced the versions of NaïveMCTS that use uniform and asymmetric abstractions, A1N, A2N, and A3N. In this work (i) we provide a much more detailed explanation of the proposed approaches; (ii) we test empirically a larger set of strategies for generating asymmetric abstractions; (iii) we show how GAB, SAB, and A3N benefit from searching even when there is no decision to be made in the game; (iv) we directly compare the methods introduced in both previous works in the domain of μ RTS, a full-fledged RTS game. Previously, GAB and SAB had only been tested in the simpler RTS combats.

This dissertation is organized as follows in Chapter 2 we review works related to real-time planning in RTS games as well as related to action abstraction in multi-unit extensive-form games. In Chapter 3 we define RTS games as an extensive-form game and review search algorithms used for planning in un-abstracted spaces. In Chapter 4 we describe the asymmetric abstractions we introduce in this work.

¹<https://github.com/santiontanon/microrts>

In Chapter 5 we introduce four novel algorithms for searching in asymmetrically-abstracted action spaces. We evaluate the algorithms we introduce in Chapter 6 with an extensive set of experiments on μ RTS.

Chapter 2

Related Work

The action abstraction scheme and the search algorithms we present in this dissertation are closely related to other search algorithms developed for RTS games. Our work is also related to learning-based systems for RTS games and to action abstraction schemes used in other contexts such as for approximating optimal solutions in extensive-form games such as Poker. In this chapter we review each of these lines of research.

2.1 Search-Based Approaches for RTS Games

Before the invention of action abstractions induced by scripts, state-of-the-art algorithms included search methods for un-abstracted spaces such as Monte Carlo [8, 9, 10, 3] and Alpha-Beta [11]. Due to the large number of actions available, Alpha-Beta and Monte Carlo methods tend to perform well only in small scale games.

In addition to PGS [1], Justesen et al. (2014) proposed two variations of UCT [12] for searching in uniformly-abstracted trees: script-based and cluster-based UCT. Wang et al. (2016) introduced Portfolio Online Evolution (POE) a local search algorithm also designed for uniformly-abstracted trees. Wang et al. showed empirically that POE is able to outperform Justesen’s algorithms, and Lelis (2017) showed that PGS and SSS are able to outperform POE. All these empirical results were obtained in a testbed named SparCraft. SparCraft is an efficient combat simulator for the combat scenarios that arise in the commercial game of StarCraft.

Justesen et al.’s and Wang et al.’s algorithms can also be modified to search in asymmetrically abstracted trees. The reason we use PGS and SSS instead of their algorithms is because PGS and SSS were shown to perform better in SparCraft [2].

A search algorithm for uniformly-abstracted trees is the core of the artificial player of the commercial card game Prismata [13]. The idea we introduce in this dissertation of performing search in asymmetric trees can also be used in such card games to enhance the strength of their artificial player. For example, the artificial player could benefit from an algorithm that discovers finer plans for the “more important” cards.

Scripts have also been used to guide the search by means other than action abstractions. Puppet Search (PS) [14] defines a search space over the parameter values of scripts. Strategy Tactics (STT) [15] combines PS’s search in the script-parameter space with a NaïveMCTS search in the original state space for the combat units. Strategy Creation via Voting (SCV) generates scripts via voting [16], which are then used to play the game. In contrast with PS, STT, and SCV that generate novel scripts during the game, the algorithms we introduce in this work use a set of scripts to generate action abstractions.

Although we evaluate our abstraction schemes by introducing variants of PGS, SSS, and NaïveMCTS. Our schemes can also be used to reduce the action space of other CMAB-based algorithms such as 2-Phase-CMAB approaches [17] and of algorithms that use enhancements such as Hierarchical Expansion [18].

2.2 Learning-Based Approaches for RTS Games

Another line of research uses learning to control units in RTS games. Search algorithms need an efficient forward model of the game to plan. By contrast, learning approaches do not necessarily require such a model. Examples of learning approaches to unit control include the work by Usunier et al. (2016) and Liu et al. (2016). Likely due to the use of an efficient forward model, search algorithms tend to scale more easily to large-scale combat scenarios than learning-based methods. While the former can effectively handle battles with more than 100 units, the latter are usually tested on battles with no more than 50 units.

The exception in terms of both scalability and strength is AlphaStar [19], the first system able to defeat professional players in the commercial game of StarCraft II. Although AlphaStar represents a landmark result in artificial intelligence research, its approach might not be easily applicable in other scenarios. This is because AlphaStar uses a large set of matches played by human players to train an initial version of the system which is then improved through the course of several days in a self-training scheme. Moreover, AlphaStar requires several TPUs and CPUs to

effectively train its agents. As mentioned above, the search-based algorithms we introduce and evaluate in this dissertation have the disadvantage of requiring an efficient forward model of the world (in our case a computer game) to perform search. Their advantage over learning-based systems such as AlphaStar is that they do not require a large set of human play data and several days of learning. The search algorithms we introduce they can be used without training for searching. Moreover, all algorithms we introduce were tested on a single CPU. Finally, the ideas we present in this dissertation are orthogonal to those presented in learning-based papers as one can combine learning algorithms with search algorithms to further improve the state of art.

2.3 Action Abstractions in Extensive-Form Games

State-space and action abstractions have also been applied to imperfect-information extensive-form games, all with applications in computer Poker [20, 21, 22, 23]. Although simultaneous-moves games such as the ones we deal with in this work can also be modeled as an imperfect-information extensive-form game [24], the abstraction schemes used to develop Poker playing agents differ considerably from the approaches we consider and introduce in this dissertation. The state space of RTS games are often far too large to employ the approach used in Poker playing agents, where a strategy is computed offline and stored to be consulted during the game. We derive our strategy during the game match, and in contrast with computer Poker, RTS games operate under harsh real-time constraints.

Chapter 3

Background

Here we define RTS games as an extensive-form game and concepts needed in later sections.

RTS games can be described as multi-unit zero-sum extensive-form games with simultaneous and durative actions, defined by a tuple $\nabla = (\mathcal{N}, \mathcal{S}, s_{init}, \mathcal{A}, \mathcal{B}, \mathcal{R}, \mathcal{T})$, where,

- $\mathcal{N} = \{i, -i\}$ is the set of **players** (i is the player we control and $-i$ is our opponent).
- $\mathcal{S} = \mathcal{D} \cup \mathcal{F}$ is the set of **states**, where \mathcal{D} denotes the set of **non-terminal states** and \mathcal{F} the set of **terminal states**. Every state $s \in \mathcal{S}$ defines a joint set of **units** $\mathcal{U}^s = \mathcal{U}_i^s \cup \mathcal{U}_{-i}^s$, for players i and $-i$. Every unit $u \in \mathcal{U}^s$ has properties that are specific to the game modeled. For example, a state could include properties such a unit's x and y coordinates on the game's map, the unit's attack range, attack damage, and hit points. $s_{init} \in \mathcal{D}$ is the start state of the game.
- $\mathcal{A} = \mathcal{A}_i \times \mathcal{A}_{-i}$ is the set of **joint actions**. $\mathcal{A}_i(s)$ is the set of legal **actions** player i can perform at state s . Each action $a \in \mathcal{A}_i(s)$ is denoted by a vector of n **unit-actions** (m_1, \dots, m_n) , where $m_k \in a$ is the action of the k -th **ready unit** of player i . A unit is not ready if it is already performing an action (unit-actions can have different durations). We denote the set of ready units of players i and $-i$ as $\mathcal{U}_{i,r}^s$ and $\mathcal{U}_{-i,r}^s$. For $k \in \mathbb{N}^+$ we write $a[k]$ to denote the action of the k -th ready unit. Also, for unit u , we write $a[u]$ to denote the action of u in a . We denote the set of unit-actions as \mathcal{M} . We write $\mathcal{M}(s, u)$ to denote the set of legal actions of unit u at s .

- $\mathcal{B} : \mathcal{D} \rightarrow \mathcal{N}' \subseteq \mathcal{N}$ is a function that receives a state s and returns the subset of players with one or more ready units in s .
- $\mathcal{R}_i : \mathcal{F} \rightarrow \mathbb{R}$ is a **utility function** with $\mathcal{R}_i(s) = -\mathcal{R}_{-i}(s)$, for any $s \in \mathcal{F}$.
- The **transition function** $\mathcal{T} : \mathcal{S} \times \mathcal{A}_i \times \mathcal{A}_{-i} \rightarrow \mathcal{S}$ deterministically determines the successor state for a state s and the set of joint actions taken at s . Note that, since actions are durative, units might still be executing their unit-actions in a_i and a_{-i} in the state returned by $\mathcal{T}(s, a_i, a_{-i})$.

The **game tree** of ∇ is a tree rooted at s_{init} in which each node represents a state in \mathcal{S} and every edge represents a joint action in \mathcal{A} . For states $s_k, s_j \in \mathcal{S}$, there exists an outgoing edge from node representing s_k to the node representing s_j in the game tree if and only if there exists $a_i \in \mathcal{A}_i$ and $a_{-i} \in \mathcal{A}_{-i}$ such that $\mathcal{T}(s_k, a_i, a_{-i}) = s_j$. Nodes representing states in \mathcal{F} are leaf nodes. We assume all trees to be finite and denote as Ψ the **evaluation function** used by algorithms while traversing the tree. Ψ receives a state s and returns an estimate of the end-game value of s for player i . All Ψ functions we consider in this dissertation are play-out based. That is, given a state s to be evaluated, one simulates the game forward while following a fixed strategy for both players for a limited number of steps, until reaching a state s' . Then, state s' is evaluated according to a domain-dependent heuristic function and this evaluation of s' is used as the estimated end-game value of s .

We call a **decision point** of player i a state s in which i has at least one ready unit. In this dissertation, the search algorithm controlling the units of a player is invoked at every node n of the game tree, even if n represents a state that is not a decision point for the player.

A **player strategy** is a function $\sigma_i : \mathcal{S} \times \mathcal{A}_i \rightarrow [0, 1]$ for player i , which maps a state s and an action a to a probability value, indicating the chance of taking action a at s . A strategy profile $\sigma = (\sigma_i, \sigma_{-i})$ defines the strategy of both players. A script $\bar{\sigma}$ also defines a player strategy. Since a script maps a state s and unit u to a unit-action for u to perform in s , a strategy can be define if $\bar{\sigma}$ is used to define the unit-action of all units of a player. That way, a script defines a player action to be performed in any state of the game.

The optimal **value of the game** rooted s for player i is denoted as $V_i(s)$ and can be computed by finding a Nash Equilibrium profile for the players. Due to the problem's size and real-time constraints, it is impractical to find optimal profiles for most RTS games. State-of-the-art heuristic search approaches use abstractions to

reduce the game tree size and then derive in real time player strategies from the abstracted trees.

3.1 Search Algorithms for Un-Abstracted Trees

In this subsection we review two search algorithms used for planning in RTS games un-abstracted trees: Alpha-Beta Considering Durations (ABCD) and Naïve Monte Carlo Tree Search (NaïveMCTS). We use ABCD and NaïveMCTS as the basis for the algorithms we introduce for searching in asymmetrically-abstracted game trees.

3.1.1 Alpha-Beta Considering Durations (ABCD)

Minimax search with Alpha-Beta pruning [25] (or Alpha-Beta for short) has been successfully applied to zero-sum sequential-move games such as Chess [26]. Alpha-Beta computes the game value of a game while pruning branches of the tree that are not reached in optimal play. Due to its pruning scheme, Alpha-Beta can dramatically reduce the number of nodes expanded, compared to regular minimax search.

Kovarsky and Buro (2005) showed how Alpha-Beta can be adapted to find an approximate solution for simultaneous-move games. Once the Alpha-Beta search reaches a node in the game tree in which both players act simultaneously, then a policy π decides who acts first, with the other player choosing their action afterwards. Examples of policies π include random and alternate selections of who is to act first. The policy π transforms a simultaneous-move game into a sequential-move game, for which Alpha-Beta is suitable. The solution encountered in the transformed sequential-move game can then be applied as an approximation to the original simultaneous-move game.

Churchill et al. (2012) showed that Alpha Beta with the alternate policy defeats the same algorithm with the random policy in RTS combats. First, we present Alpha-Beta’s original textbook pseudocode, for sequential-move games. Then, we discuss how the pseudocode is to be modified to account for Kovarsky and Buro’s contributions.

Alpha-Beta is shown in Algorithm 1, which receives as input a state as the root of the tree, a maximum depth d , bound α and β , with the initial values of $-\infty$ and ∞ , respectively, and an evaluation function Ψ . The value of d limits the depth in which Alpha-Beta’s depth-first search will be performed. In practice, we use Algorithm 1 in an iterative-deepening manner, where the value of d is initially set to 1, and it is iteratively incremented by one in case the search finishes and there

Algorithm 1 Alpha-Beta

Input: State s , depth d , $\alpha = -\infty$, $\beta = \infty$, evaluation function Ψ .**Output:** An approximation of the game value of s .

```

1: if  $s \in \mathcal{Z}$  or  $d = 0$  then
2:   return  $\Psi(s)$ 
3:  $j \leftarrow \mathcal{B}(s)$ 
4: if  $j = -i$  then
5:    $M \leftarrow \infty$ 
6:   for each  $a \in \mathcal{A}_{-i}(s)$  do
7:      $M \leftarrow \min(\text{Alpha-Beta}(\mathcal{T}(s, a), d - 1, \alpha, \beta), M)$ 
8:     if  $M \leq \alpha$  then
9:       return  $M$ 
10:    $\beta = \min(\beta, M)$ 
11: if  $j = i$  then
12:    $M \leftarrow -\infty$ 
13:   for each  $a \in \mathcal{A}_i(s)$  do
14:      $M \leftarrow \max(\text{Alpha-Beta}(\mathcal{T}(s, a), d - 1, \alpha, \beta), M)$ 
15:     if  $M \geq \beta$  then
16:       return  $M$ 
17:    $\alpha = \max(\alpha, M)$ 
18: return  $M$ 

```

is still time available for planning. Variables α and β store the best values that can be achieved by players i and $-i$, respectively. The evaluation function Ψ estimates the end-game value of a state $s \in \mathcal{D}$; $\Psi(s) = \mathcal{R}_i(s)$, if $s \in \mathcal{F}$.

As shown in Algorithm 1, the Ψ -value of a node is returned if the search reaches its maximum depth ($d = 0$ as d is decremented in each recursive call) or if the node is terminal (see line 2). Variable j stores the player who is to act in the current state (line 3), which is either i or $-i$ for sequential-move games. If it is $-i$'s turn (see lines 4–10), then Alpha-Beta searches for all possible transitions from state s , which is given by the actions in \mathcal{A}_{-i} . Note that, in the recursive call in line 7, the transition function \mathcal{T} takes two arguments: the current state s and action a . This is because in Algorithm 1 we assume sequential games, thus the transition function depends on the action of only one player. The search is pruned in $-i$'s turn (see lines 8 and 9) if the current lower bound for i 's solution value (given by α) is at least as large as the current upper bound for $-i$'s solution (given by M). If the Boolean expression $M \leq \alpha$ in line 8 is true, then it means that player i prefers to choose an earlier action in the game tree that will guarantee i a game value of α , than allow $-i$ to reach the current node of the tree, which is guaranteed to be at most as good as α . The same reasoning applies in lines 11–17, where player i is to act, instead of

player $-i$.

Algorithm 1 assumes sequential-moves games. In RTS games there are states s in which both players can act simultaneously (i.e., $\mathcal{B}(s) = \{i, -i\}$) and states s in which only one of the player acts (either $\mathcal{B}(s) = \{i\}$ or $\mathcal{B}(s) = \{-i\}$). Churchill et al. (2012) showed that Alpha-Beta can be modified to account for simultaneous-moves. Churchill et al.’s Alpha-Beta variant is known as Alpha-Beta Considering Durations (ABCD).

We need to perform two modifications in the pseudocode shown in Algorithm 1 to transform Alpha-Beta into ABCD. First, we replace command $j \leftarrow \mathcal{B}(s)$ by $j \leftarrow \pi(s)$ (line 3), where π is the policy that decides the player who is to choose their action first. For states s where $\mathcal{B}(s) = \{i\}$ or $\mathcal{B}(s) = \{-i\}$, then $\pi(s) = \mathcal{B}(s)$. If $\mathcal{B}(s) = \{i, -i\}$, then π decides which player acts first. The implementation we use employs the alternate policy: π returns i if it returned $-i$ in the previous state in which both players acted simultaneously. π randomly chooses either i or $-i$ for the first state in the tree with simultaneous actions.

The second modification deals with the “delayed effects” of an action. Since the game is artificially serialized by the search algorithm, in a state s that the players are to act simultaneously, the set of actions \mathcal{A}_{-i} (resp. \mathcal{A}_i) should be computed before applying the action player i (resp. $-i$) is going to perform at s . ABCD handles this as follows. In states with simultaneous actions, the transition function is not applied during the function’s recursive calls, as shown in lines 7 and 14 of Algorithm 1. Instead, we pass as parameters the current state s and the action a the player is going to perform at s . This way the set of actions of the other player can be computed within the next function call and only then a is applied to s . See the paper by Churchill et al. (2012) for a pseudocode of ABCD. The ABCD implementation we use in our experiments employ a transposition table [27, 28].

3.1.2 Naïve Monte Carlo Tree Search (NaïveMCTS)

Ontañón (2017) modeled the search problem of deriving strategies in RTS games as a combinatorial multi-armed bandits (CMAB) problem. A CMAB problem can be defined by a tuple (X, μ) , where,

- $X = \{X_1, \dots, X_n\}$, where each X_i is a variable that can assume K_i different values $\mathcal{X}_i = \{v_i^1, \dots, v_i^{K_i}\}$, with $\mathcal{X} = \{(v_1, \dots, v_n) \in \mathcal{X}_1 \times \dots \times \mathcal{X}_n\}$ being the possible combinations of value assignments for the variables in X ; a value assignment $V \in \mathcal{X}$ is called a macro-arm.

- $\mu : \mathcal{X} \rightarrow \mathbb{R}$ is a reward function, that receives a macro-arm and returns a reward value for that macro-arm.

The goal in a CMAB problem is to find a macro-arm that maximizes the expected reward. This can be achieved by balancing exploration and exploitation until converging to an optimal macro-arm. In the context of RTS games, each decision point s can be cast as a CMAB problem in which X contains one variable for each ready unit of a player in s . Thus, a macro-arm $V \in \mathcal{X}$ represents a player action and each value $v \in V$ represents a unit-action. The set $\mathcal{X}_i = \{v_i^1, \dots, v_i^{K_i}\}$ represents the set of K_i legal actions for the i -th unit at s . Naturally, the goal is to find a macro-arm (player action) that maximizes the player's reward, which is defined by an evaluation function.

Since the number of macro-arms in \mathcal{X} is often too large in RTS games, Ontañón (2017) derived a sampling procedure called Naïve Sampling (NS) to help deciding which macro-arms should be evaluated during search. NS divides a CMAB problem with n variables into $n + 1$ multi-armed bandit (MAB) problems.

- n local MABs, one for each variable $X_i \in X$. That is, for variable X_i representing the i -th unit, the arms of the MAB are the K_i values (unit-actions) in \mathcal{X}_i .
- 1 global MAB, denoted MAB_g , that treats each macro-arm V considered by NS as an arm in MAB_g . Naturally, MAB_g has no arms in the beginning of NS's procedure.

At each iteration, NS uses a policy π_0 to determine whether it adds an arm to MAB_g through the local MABs (explore) or evaluates an existing arm in MAB_g (exploit).

1. If explore is chosen, then a macro-arm V is added to MAB_g by using a policy π_l to independently choose a value for each variable in X . Here, NS assumes that the reward of a macro-arm V can be approximated by the sum of the rewards of the individual values $v_i \in V$, denoted $\mu'(v_i)$. That is, $\mu(V) \approx \sum_{v_i \in V} \mu'(v_i)$.
2. If exploit is chosen, then a policy π_g is used to select an existing macro-arm in MAB_g .

Ontañón (2017) showed that NS can be used in the context of Monte Carlo Tree Search (MCTS) by introducing an algorithm named NaïveMCTS (see Algorithm 2).

Algorithm 2 NaïveMCTS

Input: State s , sampling strategies π_0 , π_l and π_g , and evaluation function Ψ .**Output:** Action a

```

1: root  $\leftarrow$  node( $s$ )
2: while hasTime() do
3:   leaf  $\leftarrow$  SelectAndExpandNode(root,  $\pi_0$ ,  $\pi_l$ ,  $\pi_g$ )
4:    $v \leftarrow \Psi(\text{leaf.state})$ 
5:   propagateEvaluation(leaf,  $v$ )
6: return getMostVisitedAction(root)

```

Algorithm 3 SelectAndExpandNode

Input: A game tree node n_0 and sampling strategies π_0 , π_l and π_g **Output:** A node in the tree

```

1:  $j \leftarrow \pi(n_0.\text{state})$ 
2:  $n \leftarrow \text{NS}(n_0.\text{state}, \pi_0, \pi_l, \pi_g, j)$ 
3: if  $n \in n_0.\text{children}$  then
4:   return SelectAndExpandNode( $n_0.\text{child}(\alpha)$ )
5: else
6:    $n_0.\text{addChild}(n)$ 
7:   return return  $n$ 

```

NaïveMCTS differs from other MCTS algorithms in that it uses NS to decide which player actions should be evaluated in search. Instead of uniformly sampling which player action to evaluate next as in a vanilla MCTS algorithm, NaïveMCTS uses NS to select player actions composed of unit-actions that tend to yield good rewards. NaïveMCTS receives as input the current state s of the game and strategies π_0 , π_l and π_g , which are required by NS as discussed above. NaïveMCTS returns a player action a for player i to be played at state s .

NaïveMCTS expands a tree in its search procedure, which we will refer to as the MCTS tree. The MCTS tree starts only with the root node representing the current state of game (line 1). We denote the state that is represented by a node in the tree with the word “state” preceded by the name of the node and a dot (e.g., root.state). While there is time allowed for planning, NaïveMCTS iteratively selects a node to be added to the MCTS tree, through a call to `SelectAndExpandNode` (line 3), which is described in Algorithm 3.

Since `SelectAndExpandNode` uses the NS procedure described above, in addition to the root of the tree, it requires as input the policies π_0 , π_l , and π_g . The procedure returns a node that is added to the NaïveMCTS tree. Similarly to ABCD, NaïveMCTS uses a policy π to sequentialize simultaneous-move states by allowing one of the players to decide their action before the other player (line 1 of Algo-

rithm 3). In contrast with ABCD, NaïveMCTS ignores the possible delayed effects of actions, discussed in Section 3.1.1, caused by its artificial serialization of the game. The NS procedure is invoked to decide which player action will be explored. That is, given current state $n_0.state$ and the player j to act at $n_0.state$, which is enforced by π to be either i or $-i$, but never both, NS returns a node n whose state $n.state$ is reached by applying a player action (macro-arm) to $n_0.state$.

In line 3 the procedure checks if the node n returned by NS is already part of the MCTS tree (i.e., if n is amongst the children of n in the MCTS tree). Node n is part of the tree if NS chooses to exploit an existing macro-arm. In this case, `SelectAndExpandNode` is called recursively so that NS is invoked to select a player action from n . If NS chooses to explore, then a new macro-arm is chosen, leading to a state n that is not in the MCTS tree. In this case, n is added to the MCTS tree (line 6) and is returned to NaïveMCTS's main procedure (Algorithm 2). Although unlikely due to the large number of different macro-arms, NS's policy might choose to explore and yet choose a node that is already in the MCTS tree. In its main procedure, now under the name of *leaf*, the newly added node is evaluated with a play-out based function Ψ , and the value v returned is used as the reward value for all macro-arms (player actions) traversed from root of the MCTS tree to the leaf node added by `SelectAndExpandNode`. This is performed by procedure `PropagateEvaluation` (line 5), whose implementation is omitted in the interest of simplicity. Once NaïveMCTS runs out of time, it returns the most visited player action from $root.state$, given by a call to procedure `getMostVisitedAction`, whose implementation is also omitted (line 6).

3.2 Search Algorithms for Uniformly-Abstracted Trees

In this section we present Portfolio Greedy Search (PGS) and Stratified Strategy Selection (SSS), two algorithms developed for searching in uniformly-abstracted trees. Before presenting PGS and SSS we discuss uniform abstractions generated by a set of scripts.

3.2.1 Uniform Action Abstractions

We define a **uniform action abstraction** (or uniform abstraction for short) for player i as a function mapping the set of legal actions \mathcal{A}_i to a subset \mathcal{A}'_i of \mathcal{A}_i . Action

abstractions can be constructed from a set of scripts \mathcal{P} . Let the action-abstracted legal actions of unit u at state s be the actions for u that is returned by a script in \mathcal{P} , defined as,

$$\mathcal{M}(s, u, \mathcal{P}) = \{\bar{\sigma}(s, u) | \bar{\sigma} \in \mathcal{P}\}.$$

Definition 1 *A uniform abstraction Φ is a function that receives a state s , a player i , and a set of scripts \mathcal{P} . Φ returns a subset of $\mathcal{A}_i(s)$ denoted $\mathcal{A}'_i(s)$. $\mathcal{A}'_i(s)$ is defined by the Cartesian product of actions in $\mathcal{M}(s, u, \mathcal{P})$ for all u in $\mathcal{U}_{i,r}^s$, where $\mathcal{U}_{i,r}^s$ is the set of ready units of i in s .*

Algorithms using a uniform abstraction search in a game tree for which player i 's legal actions are limited to $\mathcal{A}'_i(s)$ for all s . This way, algorithms focus their search on actions deemed as promising by the scripts in \mathcal{P} , as the actions in $\mathcal{A}'_i(s)$ are composed of unit-actions returned by the scripts in \mathcal{P} .

3.2.2 Portfolio Greedy Search

Churchill and Buro (2013) introduced Portfolio Greedy Search (PGS), a hill-climbing search procedure for uniformly-abstracted trees. Algorithm 4 shows the pseudocode of PGS, which receives as input a state s , a default script $\bar{\sigma}_d$, a set of scripts \mathcal{P} , a time limit e , and an evaluation function Ψ . PGS returns an action vector a for player i to be executed in s .

PGS starts by selecting the script $\bar{\sigma}_i$ from \mathcal{P} that yields the largest Ψ -value when i executes an action composed of unit-actions computed with $\bar{\sigma}_i$, assuming that $-i$ executes an action composed of unit-actions computed with $\bar{\sigma}_d$ (line 1). The same process is executed to select $\bar{\sigma}_{-i}$, considering that i executes unit-actions computed by $\bar{\sigma}_i$ (line 2). Churchill and Buro (2013) called this initialization procedure the “seeding” of the players actions. Action vectors a_i and a_{-i} are initialized with the unit-actions computed from $\bar{\sigma}_i$ and $\bar{\sigma}_{-i}$. Once a_i and a_{-i} have been initialized, PGS iterates through all units u in $\mathcal{U}_{i,r}^s$ and tries to greedily improve the unit-action assigned to u in a_i , denoted by $a_i[u]$. Note that PGS only considers the unit-actions in the uniform abstraction, i.e., those in $\mathcal{M}(s, u, \mathcal{P})$. PGS evaluates a_i while replacing $a_i[u]$ by each of the possible unit-actions m for u . PGS keeps in a_i the action vector found during search with the largest Ψ -value. This process is repeated until PGS reaches time limit e and returns a_i (see lines 12 and 13).

Note that the action vector a_{-i} remains unchanged after its initialization (see line 4). Although in its original formulation PGS alternates between improving

Algorithm 4 Portfolio Greedy Search

Input: state s , default script $\bar{\sigma}_d$, set of scripts \mathcal{P} , time limit e , and evaluation function Ψ .

Output: action a for player i 's units.

```

1:  $\bar{\sigma}_i \leftarrow$  choose a script from  $\mathcal{P}$  considering that  $-i$  acts according to  $\bar{\sigma}_d$  //see text

2:  $\bar{\sigma}_{-i} \leftarrow$  choose a script from  $\mathcal{P}$  considering that  $i$  acts according to  $\bar{\sigma}_i$  //see text
3:  $a_i \leftarrow \{\bar{\sigma}_i(u_1), \bar{\sigma}_i(u_2), \dots, \bar{\sigma}_i(u_n)\}$ , where  $u_1, u_2, \dots, u_n \in \mathcal{U}_{i,r}^s$ 
4:  $a_{-i} \leftarrow \{\bar{\sigma}_{-i}(u_1), \bar{\sigma}_{-i}(u_2), \dots, \bar{\sigma}_{-i}(u_m)\}$ , where  $u_1, u_2, \dots, u_m \in \mathcal{U}_{-i,r}^s$ 
5: while time elapsed is not larger than  $e$  do
6:   for each  $u \in \mathcal{U}_{i,r}^s$  do
7:     for each  $\bar{\sigma} \in \mathcal{P}$  do
8:        $a'_i \leftarrow a_i$ ;  $a'_i[u] \leftarrow \bar{\sigma}(s, u)$ 
9:       if  $\Psi(\mathcal{T}(s, a'_i, a_{-i})) > \Psi(\mathcal{T}(s, a_i, a_{-i}))$  then
10:         $a_i \leftarrow a'_i$ 
11:   if time elapsed is larger than  $e$  then
12:     return  $a_i$ 
13: return  $a_i$ 

```

player i 's and player $-i$'s action vectors [1], in their experiments, Churchill and Buro allow PGS to improve only player i 's action vector while player $-i$'s is fixed. Moraes, Mariño, and Lelis (2018b) showed that, if set to improve both a_i and a_{-i} , PGS can suffer from a pathological issue they called the non-convergence problem. Due to the non-convergence problem, PGS can find worse strategies than PGS with a_{-i} fixed, even if the former is granted more computation time than the latter. In addition to identifying the pathology, Moraes et al.s introduced an algorithm called Nested-Greedy Search (NGS) that alters both a_i and a_{-i} while not suffering from the pathology. We use PGS with a_{-i} fixed instead of NGS because NGS was shown to not scale well to large games [29].

Another difference between our implementation of PGS and its original formulation is that we allow it to search while there is time available (see while loop in Algorithm 4). In its original formulation, PGS performs a fixed number of iterations of the while loop; Churchill and Buro allowed PGS to perform only one iteration in their experiments.

3.2.3 Stratified Strategy Selection

Lelis (2017) introduced Stratified Strategy Selection (SSS). Similarly to PGS, SSS performs a hill-climbing search. However, in contrast with PGS, SSS searches in the space of script assignments induced by a **type system**, which is a partition of

Algorithm 5 Stratified Strategy Selection

Input: state s , default script $\bar{\sigma}_d$, set of scripts \mathcal{P} , time limit e , evaluation function Ψ , and a type system T for the set of units \mathcal{U}_i in s .

Output: action a for player i 's units, boolean c indicating if the algorithm finished a complete iteration over all types in T

- 1: $\bar{\sigma}_i \leftarrow$ choose a script from \mathcal{P} considering that $-i$ acts according to $\bar{\sigma}_d$ //see text
- 2: $\bar{\sigma}_{-i} \leftarrow$ choose a script from \mathcal{P} considering that i acts according to $\bar{\sigma}_i$ //see text
- 3: $a_i \leftarrow \{\bar{\sigma}_i(u_1), \bar{\sigma}_i(u_2), \dots, \bar{\sigma}_i(u_n)\}$, where $u_1, u_2, \dots, u_n \in \mathcal{U}_{i,r}^s$
- 4: $a_{-i} \leftarrow \{\bar{\sigma}_{-i}(u_1), \bar{\sigma}_{-i}(u_2), \dots, \bar{\sigma}_{-i}(u_m)\}$, where $u_1, u_2, \dots, u_m \in \mathcal{U}_{-i,r}^s$
- 5: $c \leftarrow$ false
- 6: **while** time elapsed is not larger than e **do**
- 7: **for** each $t \in T$ **do**
- 8: **for** each $\bar{\sigma} \in \mathcal{P}$ **do**
- 9: $a'_i \leftarrow a_i$ with the actions of all units u of type t replaced by $\bar{\sigma}(u)$
- 10: **if** $\Psi(\mathcal{T}(s, a'_i, a_{-i})) > \Psi(\mathcal{T}(s, a_i, a_{-i}))$ **then**
- 11: $a_i \leftarrow a'_i$
- 12: **if** time elapsed is larger than e **then**
- 13: return a_i and boolean c
- 14: $c \leftarrow$ true // iterated over all types
- 15: return a_i and boolean c

units. SSS assigns the same script to units of the same type. For example, all units with low hit point values (type) move away from the battle (strategy of a script). A type system is defined as follows.

Definition 2 (Type System) Let \mathcal{U}_i be the set of player i 's units. $T = \{t_1, \dots, t_k\}$ is a type system for \mathcal{U}_i if it is a partitioning of \mathcal{U}_i . If $u \in \mathcal{U}_i$ and $t \in T$ with $u \in t$, we write $T(u) = t$.

Algorithm 5 shows the pseudocode of SSS, which receives as input the current state s , a default script $\bar{\sigma}_d$, a set of scripts \mathcal{P} , a time limit e , an evaluation function Ψ , and a type system for the units \mathcal{U}_i at state s . SSS returns a player action for i and a boolean value c indicating whether SSS was able to complete one iteration over the set of types in T . The boolean value c is used by SSS with Adaptive Type Systems (SSS+), explained below.

In implementation SSS also performs the seeding process described above for PGS. That is, SSS starts by selecting the script $\bar{\sigma}_i$ from \mathcal{P} that yields the largest Ψ -value when i executes an action composed of unit-actions computed with $\bar{\sigma}_i$, assuming that $-i$ executes an action composed of unit-actions computed with the default script $\bar{\sigma}_d$ (line 1). The same process is executed to select $\bar{\sigma}_{-i}$ considering that

i executes unit-actions computed by $\bar{\sigma}_i$ (line 2). SSS initializes actions a_i and a_{-i} with the unit-actions returned by the the scripts $\bar{\sigma}_i$ and $\bar{\sigma}_{-i}$. SSS then performs a greedy search to improve the unit-actions in a_i (lines 6–11). Namely, SSS evaluates all possible assignments of unit-actions according to the scripts in \mathcal{P} to units of a given type t while the unit-actions of units with types other than t are fixed. SSS keeps in a_i the unit-actions with largest Ψ -value encountered during search (lines 10 and 11). SSS returns the player action a_i once it reaches the time limit e (lines 13 and 15). Similarly to PGS, the action a_{-i} is fixed throughout search and SSS is trying to approximate a best response to the opponent’s action given by script $\bar{\sigma}_{-i}$.

Depending on the number and on the diversity of units (e.g., units with different attack ranges) present in the match, SSS might be unable to iterate through all types in T before reaching time limit e . This is because a large diversity of units result in more types being considered during search. If SSS is unable to iterate through all types, then it returns the unit-actions computed from script $\bar{\sigma}_i$ for units u whose type $T(u)$ was not accounted for during search. The assignment of $\bar{\sigma}_i$ might lead to a poor overall strategy as there could be better scripts that were not verified by the algorithm due to the lack of time. Aiming at preventing SSS from not iterating at least once over all types, Lelis (2017) developed a meta-reasoning system to adjust the granularity of the type system used. This adjustment occurs in between searches and is based on the estimated running time of a SSS iteration.

Instead of receiving one type system T , SSS receives a set of type systems with different granularities (i.e., different sizes). If the algorithm returns false with its finest type system (the one with the largest number of types), then in the next decision point it uses a coarser type system. If the algorithm returns true while using a type system T and it estimates that it can complete an iteration with a finer type system T' , then it switches to using T' .

Chapter 4

Asymmetric Action Abstractions

Uniform abstractions are restrictive in the sense that all units have their legal actions reduced to those specified by scripts. In this chapter we introduce an abstraction scheme we call **asymmetric action abstractions** (or asymmetric abstractions for short) that is not as restrictive as uniform abstractions but still uses the guidance of the scripts for selecting a subset of promising actions. The key idea behind asymmetric abstractions is to reduce the number of legal actions of only a subset of the units controlled by player i ; the sets of legal actions of the other units remain unchanged. We call the subset of units that do not have their set of legal actions reduced the **unrestricted units**; the complement of the unrestricted units are defined as the **restricted units**.

Definition 3 *An asymmetric abstraction Ω is a function receiving as input a state s , a player i , a set of unrestricted units $\mathcal{U}'_i \subseteq \mathcal{U}_{i,r}^s$, and a set of scripts \mathcal{P} . Ω returns a subset of actions of $\mathcal{A}_i(s)$, denoted $\mathcal{A}''_i(s)$, defined by the Cartesian product of the unit-actions in $\mathcal{M}(s, u, \mathcal{P})$ for all u in $\mathcal{U}_{i,r}^s \setminus \mathcal{U}'_i$ and of unit-actions $\mathcal{M}(s, u')$ for all u' in \mathcal{U}'_i .*

Algorithms using an asymmetric abstraction Ω search in a game tree for which player i 's legal actions are limited to $\mathcal{A}''_i(s)$ for all s . If the set of unrestricted units is equal to the set of units controlled by the player, then the asymmetrically-abstracted tree is equivalent to the un-abstracted tree, and if the set of unrestricted units is empty, the asymmetrically-abstracted tree is equivalent to the uniformly-abstracted tree induced by the same set of scripts. Asymmetric abstractions allow us to explore action abstractions in the spectrum of possibilities between the uniformly-abstracted and un-abstracted game trees.

Asymmetric abstractions allow search algorithms to divide its “attention” differently among the units at a given state of the game. That is, depending on the game state, some units might be more important than others (e.g., units with low hit points trying to survive), and asymmetric abstractions allow one to derive finer strategies to these units by accounting for a larger set of unit-actions for them. Similarly, a robotic control system might benefit from deriving finer plans to actuators that are more important at given state of the world. For example, the actuators controlling the arms of a robot trying to open a door are more likely to benefit from a finer plan than the actuators controlling the wheels of the robot. Uniform abstractions does not allow the planning system to “pay more attention” to specific parts of the system as they divide the search effort equally amongst all parts.

In addition to the ability of unevenly dividing the search effort amongst different parts of a system at a given state, the optimal strategy derived from an asymmetrically-abstracted tree is guaranteed to be no worse than the strategies derived from a uniformly-abstracted tree for a game ∇ , given that both abstractions are induced by the same set of scripts.

Theorem 1 *Let Φ be a uniform abstraction and Ω be an asymmetric abstraction, both defined with the same set of scripts \mathcal{P} . For a zero-sum extensive-form game ∇ with start state s , let $V_i^\Phi(s)$ be the optimal value of the game computed by considering the game tree induced by Φ ; define $V_i^\Omega(s)$ analogously. We have that $V_i^\Omega(s) \geq V_i^\Phi(s)$.*

The proof for Theorem 1 (see Appendix) hinges on the fact that a player searching with Ω has access to more actions than a player searching with Φ . This guarantee can also be achieved by enlarging the set \mathcal{P} used to induce Φ . The problem of enlarging \mathcal{P} is that new scripts might not be readily available as they need to be either handcrafted or learned. By contrast, one can easily create a wide range of asymmetric abstractions by simply modifying the set of unrestricted units. Further, in contrast with asymmetric abstractions, such an enlargement scheme would not allow one to directly focus on important parts of the system.

Although Theorem 1 guarantees that one is able to derive strategies with asymmetric abstractions that are no worse than those derived with uniform abstractions, our theory is not necessarily a good predictor of what happens in practice. This is because Theorem 1 assumes that optimal strategies can be derived from the abstracted game trees and in practice we use search algorithms to only approximate optimal solutions in real time.

In the next chapter we introduce four novel algorithms that search in real time in asymmetrically-abstracted game trees.

Chapter 5

Searching in Asymmetrically-Abstracted Game Trees

We introduce Greedy Alpha-Beta Search (GAB) and Stratified Alpha-Beta Search (SAB), two algorithms for searching in asymmetrically-abstracted trees. GAB and SAB hinge on a property of PGS and SSS that has hitherto been overlooked. Namely, both PGS and SSS may come to an early termination if they encounter a local maximum. PGS and SSS reach a local maximum when they complete all iterations of the outer for loops in Algorithms 4 and 5 without altering a_i . Once a local maximum is reached, PGS and SSS are unable to further improve the unit-action assignments, even if the time limit e was not reached.

GAB and SAB take advantage of PGS's and SSS's early termination by operating in two steps. In the first step GAB and SAB search for an action in the uniformly-abstracted tree with PGS and SSS, respectively. The first step finishes either when (i) the time limit is reached or (ii) a local maximum is encountered. In the second step, which is run only if the first step finishes by encountering a local maximum, GAB and SAB fix the moves of all restricted units according to the moves found in the first step, and search in the asymmetrically-abstracted tree for moves for all unrestricted units. If the first step finishes by reaching the time limit, GAB and SAB return the action determined in the first step. GAB and SAB behave exactly like PGS and SSS in decision points in which the first step uses all time allowed for planning. We explain GAB and SAB in more detail below.

5.1 Greedy and Stratified Alpha-Beta Searches (GAB and SAB)

In its first step GAB uses PGS to search in a uniformly abstracted space induced by \mathcal{P} for deriving an action a that is used to fix the actions of the restricted units during the second search. In its second step, GAB uses a variant of ABCD. Although we use ABCD, one could also use other search algorithms such as UCTCD [1]. ABCD is used to search in a tree we call **Move-Fixed Tree (MFT)**. The following example illustrates how the MFT is defined; MFT's definition follows the example.

Example 1 Let $\mathcal{U}_{i,r}^s = \{u_1, u_2, u_3\}$ be i 's ready units in s , $\mathcal{P} = \{\bar{\sigma}_1, \bar{\sigma}_2\}$ be a set of scripts, and $\{u_1, u_3\}$ be the unrestricted units. Let $a = (W, L, R)$ be the player action returned by PGS, where W, L, R are the unit-actions 'wait' (W), 'move left' (L), and 'move right' (R). Also, let $\varsigma = \{\bar{\sigma}_1, \bar{\sigma}_2, \bar{\sigma}_1\}$ be the script vector that defined the unit-actions during PGS's search. That is, $\bar{\sigma}_1(s, u_1) = W$, $\bar{\sigma}_2(s, u_2) = L$, and $\bar{\sigma}_1(s, u_3) = R$. We use the notation $\varsigma[u_1]$ to denote the script in ς used to define the unit-action for unit u_1 in PGS's search.

GAB's second step searches in the MFT. The MFT is rooted at s , and the set of abstracted legal player actions in s is obtained by fixing $a[u_2] = L$ and considering all legal actions for u_1 and u_3 . That is, if $\mathcal{M}(s, u_1) = \{W, U\}$ and $\mathcal{M}(s, u_3) = \{R, D\}$, then the set of abstracted legal player actions in s is: $\{(W, L, R), (W, L, D), (U, L, R), (U, L, D)\}$.

For player i and for all descendants states s' of s in the MFT, if $\mathcal{M}(s', u_1) = \{W, U\}$, $\mathcal{M}(s', u_3) = \{R, D\}$, and $\varsigma[u_2] = \bar{\sigma}_2$, then the set of abstracted legal actions in s' is:

$$\{(W, \bar{\sigma}_2(s', u_2), R), (W, \bar{\sigma}_2(s', u_2), D), \\ (U, \bar{\sigma}_2(s', u_2), R), (U, \bar{\sigma}_2(s', u_2), D)\}.$$

That is, at states s' we consider all legal unit-actions of the unrestricted units and we fix the unit-actions of the restricted units to what is returned by the units' script in ς .

Also, for all descendants states s' of s in the MFT, if player $-i$'s ready units in s are $\mathcal{U}_{-i,r}^s = \{u_1, u_2, u_3, u_4\}$, the set of abstracted legal player actions for $-i$ in s' is,

$$\{(\bar{\sigma}_{-i}(s', u_1), \bar{\sigma}_{-i}(s', u_2), \bar{\sigma}_{-i}(s', u_3), \bar{\sigma}_{-i}(s', u_4))\}.$$

Here, $\bar{\sigma}_{-i} \in \mathcal{P}$ is the script computed in PGS's seeding process (see line 2 of Algorithm 4).

Definition 4 (Move-Fixed Tree) For a given state s , a subset of unrestricted units of \mathcal{U}_i in s , a set of scripts \mathcal{P} , the script $\bar{\sigma}_{-i} \in \mathcal{P}$ defined in the seeding process of the algorithm's first step, a player action a returned by the algorithm's first step, and the script vector ς with one script for each u in $\mathcal{U}_{i,r}^s$ used by the algorithm's first step to define the unit-actions in a , a Move-Fixed Tree (**MFT**) is a tree rooted at s with the following properties.

1. The set of abstracted legal actions for player i at the root s of the **MFT** is limited to actions a' that have unit-actions $a'[u]$ fixed to $a[u]$, for all restricted units u ;
2. The set of abstracted legal actions for player i at states s' descendants of s is limited to actions a' that have unit-actions $a'[u]$ fixed to $\bar{\sigma}(s', u)$ with $\bar{\sigma} = \varsigma[u]$, for all restricted units u ;
3. The only abstracted legal action for player $-i$ at any state in the **MFT** is defined by fixing player $-i$'s unit-actions to those returned by $\bar{\sigma}_{-i}$.

By searching in the **MFT**, ABCD searches for actions for the unrestricted units while the actions of all other units, including the opponent's units, are fixed: player i 's restricted units act according to the scripts in ς and player $-i$'s units act according to $\bar{\sigma}_{-i}$. Our two-step search approximates a best response to the strategy defined by the script $\bar{\sigma}_{-i}$. In theory, this approach could make our player exploitable. However, in practice, due to the real-time constraints, one tends to derive more effective strategies by fixing the opponent strategy, as shown in previous works [29].

In our original version of GAB and SAB we defined the **MFT** differently; c.f. Example 1 and Definition 3 of Moraes and Lelis (2018). In our original work we used a script known as NOKAV in lieu of the scripts in ς and of the script $\bar{\sigma}_{-i}$ defining the opponent model. This is because the **MFT** was defined in the context of Sparcraft, a domain for which NOKAV is strong [1]. Our current definition is more general because it does not assume the existence of a strong script and it uses byproducts of the search performed in the first step (the scripts ς and $\bar{\sigma}_{-i}$) to define the **MFT**.

Let a_1 be the player action returned by PGS in GAB's first step and a_2 be the player action returned by ABCD in GAB's second step. Also, let a' be

the opponent action defined by using the script $\bar{\sigma}_{-i}$ for all opponent's units. Instead of returning a_2 directly, GAB returns the action with largest Ψ value, i.e., $\arg \max_{a \in \{a_1, a_2\}} \Psi(\mathcal{T}(s, a, a'))$. Note that one cannot compare the evaluation value of actions a_1 and a_2 as computed by ABCD and PGS. This is because ABCD performs a depth-first search and uses the Ψ function to evaluate the leaf nodes of the tree expanded by ABCD; these values are then propagated up the tree to evaluate the actions available at the root. As a result, the evaluation values of the actions at the root performed by ABCD are based on nodes deeper into the tree than the evaluation performed by PGS. Thus, once GAB has a_1 and a_2 , it evaluates them again with the same Ψ function, and only then it selects the best of the two actions.

The difference between SAB and GAB is the algorithm used in their first step: while GAB uses PGS, SAB uses SSS. The second step of SAB follows exactly GAB's second step.

5.1.1 Searching in Within-States

The transition function \mathcal{T} receives as input the current state, a set of joint player actions and returns the next state in the game. Since the unit-actions are durative, the state s' returned by \mathcal{T} might not be a decision point for either of the players (i.e., none of the players have a ready unit). We call the states s that are not a decision point for player i a **within-state** for i . In RTS games, within-states are those in which the game shows an animation of the units performing their actions (e.g., a worker building a structure) and the player cannot issue an action. More generally, within-states occur when an agent is performing a predefined sequence of actions—e.g., a macro-action [30, 31, 32] or an option [33, 34]—and for that the agent is not ready to perform another action.

GAB and SAB are implemented to take advantage of within-states. This is achieved by performing the first step of GAB and SAB in within-states and the second step in decision points. While the agent is performing a sequence of actions (e.g., unit building a structure) at within-states s' for player i , GAB and SAB use the game model to fast forward from s' to a decision point s for i . Since one might encounter decision points for player $-i$ while fast forwarding, GAB and SAB assume the opponent follows the strategy given by the script selected to initialize the opponent's action in PGS and SSS, $\bar{\sigma}_{-i}$. GAB and SAB's first step is then performed at s and its result stored in memory. Later, if s is reached in the actual game, then GAB and SAB performs only their second step, searching for the unrestricted units with ABCD; the restricted units perform the action stored in memory for s .

State s might not be reached in the actual game as the opponent might choose actions that are different from those returned by the opponent model defined in the first step of search. If instead of reaching s , the players reach a game state s'' and the set of units at s and s'' are identical for player i , i.e., $\mathcal{U}_i^s = \mathcal{U}_i^{s''}$, then GAB and SAB performs only the second step, leaving the actions of the restricted units fixed, as described above. However, if $\mathcal{U}_i^s \neq \mathcal{U}_i^{s''}$, then GAB and SAB perform both the first and second steps at s'' .

Searching in within-states can be helpful because the search algorithm is allowed more time to search. However, it is not clear that this search scheme allows the algorithms to encounter stronger strategies. This is because when searching in a within-state, GAB and SAB assume a model of the opponent so that the game can be fast forwarded to a decision point for player i , from which GAB and SAB search with PGS and SSS. As the game progresses, the results of such a within-state search is used to define the actions of the restricted units, even if the decision point is not reached in the actual game. This means that the search performed at the fast-forwarded decision point might have considered, for example, a set of enemy units that is different from the set of enemy units in state reached in the actual game, and these discrepancies can harm the strategy encountered by the search. We show empirically in Chapter 6.2 that the within-states search can be effective in practice.

5.1.2 Baselines for GAB and SAB: $\text{GAB}_{\mathcal{P}}$, $\text{SAB}_{\mathcal{P}}$, GAS, and SAS

In this section we introduce four baseline algorithms for GAB and SAB. The goal of introducing these baselines is show empirically the advantages of (i) searching in asymmetrically-abstracted trees and (ii) of searching with a two-step scheme. Similarly to GAB and SAB, the first two baselines we introduce, which we name $\text{GAB}_{\mathcal{P}}$, $\text{SAB}_{\mathcal{P}}$, use a two-step search scheme. However, instead of searching in asymmetrically-abstracted trees, they search in uniformly-abstracted trees. The other two baselines, which we name Greedy Asymmetric Search (GAS) and Stratified Asymmetric Search (SAS), also search in asymmetrically-abstracted trees, but instead of performing two steps, it searches in a single step. Similarly to GAB and SAB, all four baselines benefit from searching in within-states.

$\text{GAB}_{\mathcal{P}}$ and $\text{SAB}_{\mathcal{P}}$ In contrast with GAB and SAB, $\text{GAB}_{\mathcal{P}}$ and $\text{SAB}_{\mathcal{P}}$ only account for unit-actions in $\mathcal{M}(s, u, \mathcal{P})$ for all s and u in their ABCD search. That is, $\text{GAB}_{\mathcal{P}}$

and $\text{SAB}_{\mathcal{P}}$ only consider actions a' for which the unit-actions $a'[u]$ for restricted units u are fixed (as in GAB's and SAB's MFT) and the unit-actions $a'[u']$ for unrestricted units u' that are in $\mathcal{M}(s, u', \mathcal{P})$. $\text{GAB}_{\mathcal{P}}$ and $\text{SAB}_{\mathcal{P}}$ focus their search on a subset of units \mathcal{U}' by searching deeper into the game tree with ABCD for \mathcal{U}' . In addition to searching deeper with ABCD, GAB and SAB focus their search on a subset of units \mathcal{U}' by accounting for all legal moves of units in \mathcal{U}' during search. If granted enough computation time, optimal algorithms using Ω derive stronger strategies than optimal algorithms using Φ (Theorem 1). In practice, due to the real-time constraints, algorithms are unable to compute optimal strategies for most of the decision points. We analyze empirically, by comparing $\text{GAB}_{\mathcal{P}}$ to GAB and $\text{SAB}_{\mathcal{P}}$ to SAB, which abstraction scheme allows one to derive stronger strategies.

GAS and SAS The difference between GAS and PGS is that in the greedy search of the former, for a given state s , instead of limiting the number of legal actions of all units u to $\mathcal{M}(s, u, \mathcal{P})$, as PGS does, GAS considers all legal actions $\mathcal{M}(s, u)$ for unrestricted units, and the actions $\mathcal{M}(s, u, \mathcal{P})$ for restricted units. While PGS searches in a uniformly-abstracted tree, GAS searches in an asymmetrically-abstracted tree. The difference between SAS and SAB is twofold. First, similarly to the difference between GAS and PGS, SAS also accounts for actions $\mathcal{M}(s, u)$ for all unrestricted units u . Second, in the type system T used by SAS, all unrestricted units u have their own type, i.e., $T(u) \neq T(u')$ for all unrestricted units u and all units u' . This second modification is needed to guarantee that SAS is similar to SSS in that units of the same type execute the action returned by the same script. Suppose that there was an unrestricted unit u for which $T(u) = T(u')$ and SAS selected an unit-action m to be executed by u that is not returned by any of the scripts in \mathcal{P} , i.e., $m \notin \{\bar{\sigma}(u) | \bar{\sigma} \in \mathcal{P}\}$. In this case, one way of guaranteeing SSS' principle that units of the same type execute the action returned by the same script is to ensure that each unrestricted unit has its own type.

5.2 Asymmetrically Action-Abstracted NaïveMCTS (A3N)

We call Asymmetrically Action-Abstracted NaïveMCTS (A3N) the version of NaïveMCTS that accounts during search for all unit-actions of the unrestricted units and only for the actions returned by the set of scripts \mathcal{P} for the restricted

units.¹ The only different between NaïveMCTS and A3N is that in the latter, the NaïveSampling procedure (see call to NS in Algorithm 3) can only sample macro-arms that are in the asymmetrically-abstracted tree.

Similarly to the search algorithms discussed above, A3N can also benefit from searching in within-states. This is achieved by maintaining in memory the parts of the tree expanded by A3N that are still reachable in the game. If the current game state s' is a within-state to both players, then A3N fast forwards to a state s that is a decision point to either of the players and searches from s ; A3N stores in memory the tree expanded during search. If s' is a within-state only to player i , then A3N searches from s' directly as, similarly to NaïveMCTS, A3N searches for actions for both players (see Algorithm 3); A3N also stores in memory the tree expanded during search. Once the next state s'' of the actual game is reached, which could be either a within-state or a decision point to player i , A3N removes from memory the parts of the tree that are no longer reachable from s'' and makes s'' the root of the tree. Note that s'' might have been added to the tree in the search of a previous state and A3N might already have evaluated player actions (macro-arms) available at s'' . The time allowed for planning at s'' will serve to explore more macro-arms and search deeper into the tree rooted at s'' to obtain more accurate estimates of the end-game values of the macro-arms that were already added to the MAB_g of s'' .

5.2.1 Baselines for A3N: A1N and A2N

Similarly to the baselines introduced for GAB and SAB, we introduce two baselines for A3N: A1N and A2N. Both are based on NaïveMCTS, with the former searching in uniformly-abstracted trees and the latter searching in asymmetrically-abstracted trees. The goal of introducing these baselines is to allow us to evaluate empirically the effectiveness of the asymmetric abstractions introduced above for A3N in comparison to uniform abstractions induced by \mathcal{P} and asymmetric abstractions induced by two sets of scripts. A1N and A2N also use the A3N enhancement of reusing the search tree expanded in previous states.

A1N We call A1N a version of NaïveMCTS that uses an action abstraction induced by \mathcal{P} . The difference between NaïveMCTS and A1N is in the unit-actions sampled by NS while adding macro-arms to MAB_g . Instead of being able to sample from all legal unit-actions, A1N's is allowed to sample only from $M(s, u, \mathcal{P})$ for all units u .

¹The number '3' in A3N is the version of the algorithm; versions 1 and 2 (A1N and A2N) are described in Section 5.2.1.

As a consequence, the macro-arms added to MAB_g are restricted to the unit-actions returned by the scripts.

A2N We call A2N the version of NaïveMCTS that uses an action abstraction defined by two sets of scripts: \mathcal{P}' and \mathcal{P}'' . A2N divides the set of units into two subsets: the units related to \mathcal{P}' and the units related to \mathcal{P}'' . A2N can only sample unit-actions m for the units u in the first group if m is returned by one of the scripts in \mathcal{P}' for u . The unit-actions A2N can sample for the second group of units is defined analogously. Note that the two subsets of units do not need to be disjoint as some units can have actions sampled from both sets of scripts. The action abstraction used by A2N is also asymmetric as the number of scripts in each set can be different, allowing A2N to derive finer plans to units in either group.

Chapter 6

Empirical Evaluation Methodology and Settings Definition

We evaluate the algorithms proposed in this dissertation to searching in asymmetrically-abstracted action spaces on μ RTS [3], a real-time strategy game developed for research purposes; the game is detailed in Section 6.1. Our empirical evaluation is divided into four parts. First, we evaluate the effectiveness of searching in within-states (Section 6.2). Then, we evaluate different strategies for selecting the set of unrestricted units (Section 6.3). We then evaluate GAB, SAB, and A3N against their baselines GAS, $GAB_{\mathcal{P}}$, SAS, $SAB_{\mathcal{P}}$, A1N and A3N (Section 6.4). Finally, we compare GAB, SAB, and A3N against state-of-the-art search algorithms for RTS games (Section 6.5).

6.1 μ RTS

μ RTS is a challenging open-source RTS game developed for research purposes [3]. Figure 6.1 shows a screenshot of a μ RTS state of a match played on a 8×8 map. Circles and squares with a colored contour denote units that can be controlled by one of the players (blue or red). Colored squares without a contour can be either numbered or unnumbered, with the former denoting resources (the number shows the quantity of resources available) and the latter denoting obstacles that cannot be traversed by units. Uncolored squares are traversable regions of the map. The letters are not part of the game and were added to the figure to ease the description of the units and game mechanics that follows.

Unit types Each player can train the following types of units: workers, light units,

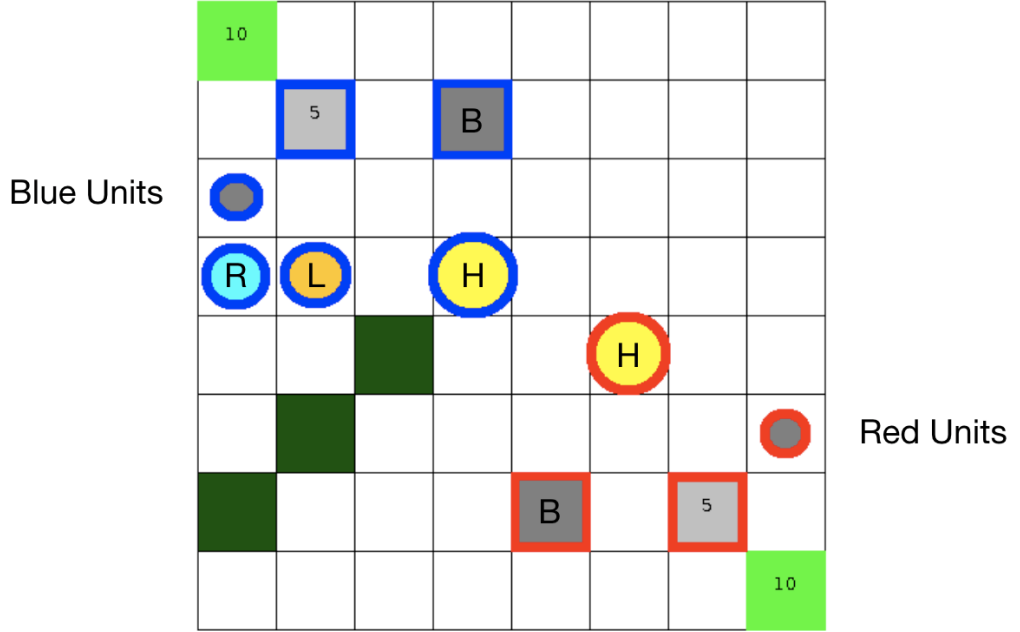


Figure 6.1. A μ RTS state of a game in a 8×8 map.

ranged units, heavy units, base, and barracks. The units that can move and attack are represented by a circle, the other units are represented by a square. The smallest circles (dark-gray) represent workers, yellow circles heavy units (with the letter 'H'), light-blue circles ranged units (with the letter 'R'), and small orange circles light units (with the letter 'L'). Light-gray squares represent bases (the number in the base shows the amount of resources available to the player). The dark-gray squares represent barracks (with the letter 'B').

Hit points Every unit has an amount of hit points that indicates the amount of damage the unit can suffer before being removed from the game. Workers and ranged units have fewer hit points than light and heavy units. The base has more hit points than any other unit. Some of the units can attack an enemy unit. If unit u attacks enemy unit u' , then u reduces the hit points of u' according to its inflicted damage, which is determined by u 's type. Workers and ranged units cause the least amount of damage, light and heavy units cause more damage. Workers, light, and heavy units u can only attack enemy units adjacent to the grid cell u occupies. Ranged units u can attack any enemy unit that is at an Euclidean distance of 3 grid cells or less from u .

Action scheme Most of the unit-actions require a single game cycle to be executed

(RTS games typically have from 10 to 50 game cycles per second [4]), but some of the unit-actions (e.g., build a base) take several game cycles to complete (i.e., actions have different durations). Any unit can take a no-op action, which means that the unit waits until the next game cycle. All units other than base and barracks can move one grid cell at a time (up, down, left, and right). Only one unit can occupy a given grid cell at a time. If two units move simultaneously to the same grid cell, then the game server overwrites their actions with the no-op action.

Collecting and spending resources Bases and barracks cannot move nor attack, but the former can train workers and the latter can train light, heavy, and ranged units—at a cost of resources. Workers can build bases and barracks. Workers can also collect resources (one unit of resource at a time). Once collected, the resource must be delivered to the base by the worker. Once the collected resource is delivered at the player’s base, it can be spent to train other units or build bases and barracks.

6.1.1 Empirical Setting for μ RTS

In μ RTS players need to submit an action at every decision point. Each player is allowed 100 milliseconds for planning in each state of the game (decision point or within-state).

Every match in our experiments is limited by several game cycles, and the match is considered a draw once the limit is reached. The maximum number of game cycles is dependent on the map. We use the limits defined by Barriga et al. [14]. The strategy played in μ RTS varies depending on the map used. For example, “rush” strategies where the player trains weak units and send them out to quickly attack the opponent tend to work better in smaller maps, while economy-based strategies where the player evolves an economy tend to work better in larger maps. Table 6.1 shows the name of the maps, their sizes, and the maximum game cycles allowed. We use 10 maps of varied sizes in our experiments, from small maps created for research, to large maps used in commercial games (BloodBathB and BloodBathD are copies of StarCraft’s BloodBath map; BloodBathB and BloodBathD differ only on the initial position of the players in the map). Each tested algorithm plays against every other algorithm ten times in each map. To ensure fairness, the players switch their starting location on the map an even number of times. For example, if Algorithm 1 starts in location X with Algorithm 2 starting in location Y for five matches, we switch the starting positions for the remaining five matches.

Map Name	Size	Number of Cycles
basesWorkers8x8A	8×8	3,000
FourBasesWorkers8x8	8×8	3,000
basesWorkers16x16A	16×16	4,000
TwoBasesBarracks16x16	16×16	4,000
basesWorkers24x24A	24×24	5,000
basesWorkers24x24ABarrack	24×24	5,000
basesWorkers32x32A	32×32	6,000
basesWorkersBarracks32x32	32×32	6,000
(4)BloodBathB	64×64	8,000
(4)BloodBathD	64×64	8,000

Table 6.1. Maps used in our experiments. The names are as they appear in the μ RTS codebase. We also show the size of the maps and the maximum number of game cycles for matches played in each map.

The evaluation function Ψ used with our algorithms is the average result of two random play-outs. A random play-out evaluates a state s by simulating the game forward from s for n game cycles with both players choosing actions randomly, until reaching a state s' . PGS, SSS, GAB, and SAB use $n = 200$, while A1N and A3N use $n = 100$. Then, the evaluation of s is given by $\Phi(s')$, where Φ is a function introduced by Ontañón (2017). Φ computes a score for each player— $score(max)$ and $score(min)$, for players i and $-i$, respectively—by summing up the cost in resources required to train each unit controlled by the player weighted by the square root of the unit’s hit points. The Φ value of a state is given by player max ’s score minus player min ’s score. Φ is then normalized to a value in $[-1, 1]$ through the following formula $\frac{2*score(max)}{score(min)+score(max)} - 1$.

The set of scripts used by PGS, SSS, GAB, SAB, and A1N is worker rush (WR), light rush (LR), heavy rush (HR), and ranged rush (RR) [35, 16]. A3N’s set of scripts is composed of LR, HR, and RR. A3N uses a different set of scripts because preliminary results showed that the algorithm tend to perform better with LR, HR, and RR. We use LR as the default script for PGS, SSS, GAB, and SAB. All these scripts train units which are immediately sent to attack the enemy. The difference among them is the type of unit trained, WR trains workers units, LR trains light units, HR trains heavy units, and RR trains ranged units. The ABCD algorithm used in GAB and SAB uses the technique called scripted move ordering to allow for more pruning during search [11]. In our experiments, the ABCD search of GAB and SAB first searches the actions returned by WR for maps of size 8×8 and 16×16 and the actions returned by LR for the other maps before considering

Algorithms	Map Size				
	8×8	16×16	24×24	32×32	64×64
GAB _w × GAB	60.0	100.0	90.0	95.0	55.0
SAB _w × SAB	55.0	90.0	70.0	70.0	80.0
A3N _w × A3N	65.0	70.0	75.0	75.0	70.0

Table 6.2. Winning rate of GAB_w, SAB_w, and A3N_w against their baselines.

other actions.

Our results will be reported in terms of winning rate. The winning rate is computed by summing the total number of victories and half of the number of draws of each algorithm evaluated and then dividing this sum by the total number of matches played; the result of the division is then multiplied by 100. For example, if an algorithm wins all matches against an opponent, then it has a winning rate of 100.0 against that opponent.

6.2 Evaluating Within-State Search

The within-states, introduced in section 5.1.1, allows GAB, SAB, and A3N to search even in states for which the player has no ready units—the within-states. One should expect that the version of A3N that searches in within-states, which we will denote as A3N_w, to outperform the version of A3N that searches only in decision points. This is because A3N_w only uses the information contained in the tree expanded while searching in within-states if that information is relevant, i.e., if the game reaches a state in the expanded tree. On the other hand, as explained in Section 5.1.1, it is not clear if the versions of GAB and SAB that search in within-states, which we denote as GAB_w and SAB_w, will outperform the versions of GAB and SAB that search only in decision points.

In this section we evaluate empirically GAB_w, SAB_w, and A3N_w against GAB, SAB, and A3N. When defining an asymmetric action abstraction one needs to define a strategy for selecting the set of unrestricted units at every state of search. In this experiment we select player *i*’s unit that is closest to an enemy unit as the only unrestricted unit. We describe and evaluate several domain-specific strategies for selecting unrestricted units in Section 6.3. Table 6.2 shows the winning rate of each within-state variant against their baseline. The results are averaged by map size. As expected, A3N_w defeats A3N on average on all map sizes tested. The within-states versions of GAB and SAB also defeat their baselines. Both GAB_w

and SAB w obtain winning rates equal or larger than 70% against their baselines in matches played on maps of size 16×16 , 24×24 , and 32×32 . The winning rates are lower for the smaller 8×8 map. This is because the strategy encoded in the script WR is already very strong in this map and both GAB and SAB play a strategy that is similar to WR. GAB w and SAB w also play a strategy similar to WR, but they are able to provide better control to the units during combat, thus improving slightly the results.

There is a discrepancy in the results for matches played in maps of size 64×64 . While SAB w defeats SAB by a large margin, it wins 80% of the matches, GAB w defeats GAB in only 55% of the matches. Due to the size of the map, GAB and SAB spend all their time allowed for planning on their first step: PGS and SSS, respectively. Thus, GAB plays identically to PGS and SAB plays identically to SSS on these maps. Also, as demonstrated by results not shown in the table, PGS is able to encounter stronger strategies than SSS for matches played on these maps. For returning stronger strategies in its first step, GAB w is unable to significantly improve the strategy with its second step. By contrast, since the strategy returned by SAB w 's first step tends to be weaker, its second step is able to substantially improve the quality of the strategy encountered. Henceforth in this dissertation, all experiments with GAB, SAB, A3N and their baselines are with the version of the algorithms that search in within-states. We drop the 'w' from the algorithms' names to ease notation.

We also collected the frequency in which the actions returned by the first steps of GAB and SAB are used in search, i.e., the frequency in which they are not disregarded because i 's unit set in the simulated state s differs from the set in the reached state s'' . The experiments are run on the 10 maps, with 20 matches in each map. The frequency results are shown in Figure 6.2, where they are averaged per map size; error bars denote standard deviation.

In general, both algorithms presented numbers above 90% of use frequency for the search results obtained in within-states. The lowest frequency is achieved on the smaller maps 8×8 and 16×16 . We hypothesize that in smaller maps the players are constantly battling each other. It is thus more common to encounter states s and s'' for which the unit set of player i differs because some of the units are removed as a result of an ongoing battle. The high frequency in which the search result of within-states is used helps explain the results shown in Table 6.2: on average, the algorithms searching in within-states have more time for planning in more than 90% of the decision points encountered during a match.

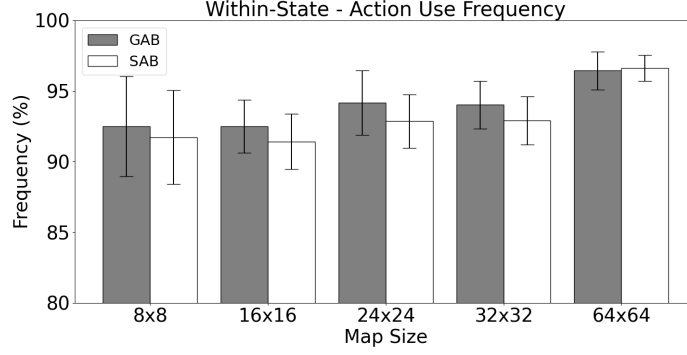


Figure 6.2. The within-state frequency of use by GAB and SAB.

6.2.1 Within-State Version of Baselines

Given the strong empirical results of the within-state version of GAB, SAB, and A3N, we have also implemented the within-state versions of all baselines. A1N and A2N are implemented in the same way as A3N, where the Monte-Carlo tree is expanded also in within-states. PGS, GAS, SSS, and SAS also make use of within-states by running their hill-climbing search at a fast-forwarded decision point s ; similarly to GAB and SAB, here we assume an opponent model to allow one to fast forward from a within-state to a decision point. If state s is indeed reached in the actual game or if the reached state s'' and s have the same unit set for player i , then PGS, GAS, SSS, and SAS continue their hill-climbing search from where it stopped in the within-state search; the algorithms search from scratch otherwise. $GAB_{\mathcal{P}}$ and $SAB_{\mathcal{P}}$ also search in within-states exactly as GAB and SAB do.

6.3 Evaluating Strategies and Number of Unrestricted Units

Next, we describe and evaluate nine strategies for selecting the unrestricted units. A selection strategy receives a state s and a set size N and returns a subset of size N of the player's units. The selection of unrestricted units is dynamic as the strategies can choose different unrestricted units at different states. Ties are broken randomly in our strategies.

1. **Farthest from Centroid (FC).** FC selects the N units that are farthest from the centroid of all player i 's units. The intuition behind FC is to allow a finer control for the units that are distant from other ally units and might be unprotected.

2. **Closest to Centroid (CC)**. CC selects the N units that are closest to the centroid of all player i 's units. This strategy serves as a baseline for FC.
3. **Closest to Enemy (CE)**. CE selects the N units that are closest to an enemy unit at every decision point. Similarly to FC, the intuition behind CE is to allow a finer control for the units that are likely to be more threatened by enemy units.
4. **Farthest from Enemy (FE)**. FE selects the N units that are the farthest from an enemy unit. The intuition is that by providing a finer control to units that are far of the enemy, one might achieve a better overall positioning of units.
5. **Less life (HP-)**. HP- selects the N units with the lowest hit points. The intuition of this strategy is to provide a finer control to units that are about to be eliminated from the game, hoping that a finer control will keep these units longer in the match.
6. **More life (HP+)**. HP+ selects the units with more hit points at a given decision point. This strategy can be helpful in scenarios where it is important to provide a finer plan to the structures responsible for training units (base and barracks) as they are the units with largest number of hit points in the game.
7. **High Attack Value (AV+)**. Let $av(u) = \frac{dpf(u)}{hp(u)}$, where $dpf(u)$ is the amount of damage per game cycle a unit can inflict to an enemy unit and $hp(u)$ is u 's current amount of hit points. AV+ selects the N units with the largest av -values. AV+ is similar to HP- as they both provide a finer control to units with low hp -value. However, AV+ selects units with low hp -value and/or large dpf -value, while HP- only accounts for hp . Moraes and Lelis (2018) showed that AV+ yields the best results for combat scenarios that arise in RTS matches.
8. **Low Attack Values (AV-)**. AV- selects the units with the lowest av -values.
9. **Random (R)**. R randomly selects N units. This strategy serves as a baseline for the other strategies.

We evaluate GAB, SAB, and A3N with the nine strategies described above for values of $N \in \{1, \dots, 10\}$. We compare each algorithm with its baseline that searches in uniformly-abstracted spaces, PGS, SSS, and A1N. Each algorithm plays against its baseline ten times in each one of the ten maps. Table 6.3 shows the

GAB vs. PGS										
Strategy	Unrestricted Set Size N									
	1	2	3	4	5	6	7	8	9	10
CC	59.0	59.5	47.0	57.0	43.0	46.0	43.0	44.0	45.5	45.5
FC	68.0	55.0	50.5	46.0	44.0	41.0	43.0	40.0	54.0	41.0
CE	67.5	68.0	62.0	60.5	59.0	53.0	50.0	43.5	49.0	56.5
FE	76.0	75.0	69.0	74.5	67.0	54.5	52.5	38.5	46.5	41.5
AV-	77.0	74.0	82.0	79.0	73.0	70.0	68.0	64.0	53.0	54.0
AV+	69.0	71.0	76.0	74.0	70.0	77.0	74.5	57.5	55.5	67.0
HP-	68.0	72.0	82.0	76.0	66.0	73.0	65.5	63.0	67.0	61.0
HP+	61.5	57.0	57.0	39.5	43.0	36.0	47.5	35.5	38.0	36.0
R	63.5	47.5	44.5	45.0	47.5	42.5	49.0	45.5	56.5	48.5

SAB vs. SSS										
Strategy	Unrestricted Set Size N									
	1	2	3	4	5	6	7	8	9	10
CC	65.0	66.5	54.0	57.0	50.0	60.0	59.0	58.0	61.0	57.0
FC	56.0	61.5	51.0	61.5	55.0	56.5	51.0	54.0	58.5	56.5
CE	72.0	60.0	56.0	60.5	54.0	56.0	55.0	51.5	55.0	48.0
FE	63.0	73.0	75.0	69.0	59.0	60.5	60.0	58.0	57.0	55.5
AV-	70.0	69.0	73.0	67.0	67.0	63.0	70.0	65.0	54.0	60.0
AV+	66.0	74.0	72.0	71.0	71.0	68.0	65.5	66.0	61.5	61.5
HP-	71.5	71.0	72.0	67.0	61.0	67.5	59.0	69.0	66.0	68.0
HP+	54.0	58.0	57.0	52.5	50.5	50.5	58.0	59.0	60.5	54.5
R	66.5	54.0	60.0	56.5	62.0	56.0	58.0	62.0	59.5	60.0

A3N vs. A1N										
Strategy	Unrestricted Set Size N									
	1	2	3	4	5	6	7	8	9	10
CC	69.6	59.2	57.1	57.5	59.2	54.2	51.3	41.3	37.5	40.0
FC	68.8	72.5	71.7	78.3	71.7	68.8	68.3	64.2	55.4	62.5
CE	72.5	69.2	75.4	71.7	77.9	75.0	72.1	63.3	57.1	57.1
FE	59.2	64.2	56.3	55.0	56.3	45.0	40.8	35.4	25.8	26.7
AV-	37.9	27.9	27.1	25.0	26.7	24.6	20.0	17.5	18.3	22.9
AV+	21.7	40.4	48.8	65.0	60.8	56.7	62.1	60.8	57.1	56.7
HP-	24.2	38.3	55.4	58.3	59.2	57.9	60.8	64.6	61.7	51.7
HP+	31.7	22.1	21.3	30.8	24.6	24.6	23.3	33.3	35.0	32.1
R	69.6	69.6	63.3	70.4	65.0	55.8	55.0	55.4	52.9	52.1

Table 6.3. Winning rate of variants of GAB against PGS in 100 matches played in 10 maps, 10 matches for each map. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

average winning rate of the algorithms for different strategies and values of N . The rows show the strategies used for selecting the unrestricted set while the columns

show the size of the set. We use a cell-coloring scheme in Table 6.3 to aid us understand the results. In our color scheme, the lowest winning rate in the table (17.5 for A3N with AV- and $N = 8$) has the lightest color and the largest winning rate (82.0 for GAB with AV- and HP- with $N = 3$) has the darkest color. The remaining cell colors are chosen as a linear interpolation of the colors of the two extremes.

All three algorithms tend to perform better with smaller values of N ($N \leq 5$). This is because for larger N the space becomes too large to allow the algorithm to encounter strong strategies under real-time constraints. Table 6.3 also shows that the algorithms searching with asymmetric action abstractions can outperform their baselines that search with uniform action abstractions. The largest winning rate obtained by GAB, SAB, and A3N are 82.0 (AV- or HP- with $N = 3$), 75.0 (FE with $N = 3$), and 78.3 (FC with $N = 4$), respectively. GAB, SAB, and A3N outperform their baselines even with the random strategy with $N = 1$. Henceforth, we use GAB with HP- and $N = 3$, SAB with FE with $N = 3$ and A3N with FC and $N = 4$. Next, we explain the results presented in Table 6.3 using domain-dependent knowledge. The reader not interested in the problem domain should skip to Section 6.4.

GAB performs best with AV- and HP-, two dissimilar strategies. Strategy AV- allows GAB to provide a finer control to bases and barracks, as these units minimize the AV-value (they are unable to cause damage and have a large number of hit points). Strategy HP- allows GAB to provide a finer control to weaker units such as workers or combat units that have suffered damage. SAB also obtains good results with both AV- and HP-. These results are in contrast with A3N's, as A3N can be worse than A1N if using either AV- and HP-.

The discrepancy in results with the AV- strategy happens because both GAB and SAB use scripts to sort the actions explored in the ABCD search. For example, in maps of size 16×16 , ABCD evaluates the action provided by the LR script before any other action. The LR strategy builds a barracks as soon as possible so that light units can be trained, and a barracks can only be built if the player "saves" resources. Due to the ABCD move ordering, both GAB and SAB are able to evaluate the sequence of actions to successfully build a barracks while using the AV- strategy. By contrast, A3N does not employ a move ordering approach and the actions needed to produce a barracks might not even be evaluated if one considers all legal actions of bases and barracks (as it happens with the AV- strategy). Moreover, A3N uses a play-out function to evaluate actions that is shorter than the one used by GAB and SAB. As a result, even if A3N evaluates the action of building a barracks, the

algorithm is shortsighted and thus unable to perceive the value of building such a structure. A3N also obtains poor results with strategy HP+ for exactly same reasons just described.

The discrepancy of results of GAB/SAB and A3N with strategy HP- can be explained by similar arguments. The HP- strategy allows the algorithms to mostly control workers, which are the units with the lowest hit point values. Workers are usually either battling the opponent or collecting resources. Due to its lack of move ordering, if providing a finer control to workers collecting resources, A3N often mistakenly sends such units to attack the opponent, thus interrupting their task. The move ordering used by GAB and SAB's ABCD search allows the algorithms to not harm the player's strategy for resource gathering.

Strategies that are strong for GAB are also strong for SAB. However, in contrast with GAB, SAB outperforms its baseline with almost any strategy and value of N . This happens likely because SAB has the weakest of the baselines. The SSS search is more limited than PGS because it is constrained to a type system. Thus, SAB's ABCD search can more easily improve upon the actions encountered by the algorithm's first step.

A3N performs best with the CE and FC strategies. Both strategies allow A3N to provide a finer control to units in direct combat with the enemy. A3N performs better with these strategies than GAB and SAB likely because A3N does not assume a model of the opponent and is thus more robust in combat scenarios. By contrast, GAB and SAB's ABCD search assume a model of the opponent and the algorithms might perform poorly if the opponent follows a strategy that is different than the one assumed during search.

6.4 Comparison with Baselines

In this section, we evaluate GAB, SAB and A3N against their baselines PGS, GAS, $GAB_{\mathcal{P}}$, SSS, SAS, $SAB_{\mathcal{P}}$, A1N and A2N. Table 6.4 shows the results for GAB and SAB, while Table 6.5 shows the results for A3N. The numbers in the tables indicate the winning rate of the row player against the column player.

All three algorithms, GAB, SAB, and A3N, outperform their baselines. The results of GAB against $GAB_{\mathcal{P}}$, SAB against $SAB_{\mathcal{P}}$, and A3N against A1N demonstrate that algorithms that search with asymmetric action abstractions can substantially outperform their counterparts that search with uniform abstractions. The results of GAB against GAS and SAB against SAS demonstrate that the two-step search

	PGS	GAS	GAB _{\mathcal{P}}	GAB	Avg.
PGS	-	78.5	84.0	19.0	60.5
GAS	21.5	-	52.0	8.0	27.2
GAB _{\mathcal{P}}	16.0	48.0	-	20.0	28.0
GAB	81.0	92.0	80.0	-	84.3
	SSS	SAS	SAB _{\mathcal{P}}	SAB	Avg.
SSS	-	85.0	73.0	26.0	61.3
SAS	15.0	-	27.5	8.0	16.8
SAB _{\mathcal{P}}	27.0	72.5	-	26.0	41.8
SAB	74.0	92.0	74.0	-	80.0

Table 6.4. Winning rate of the row player against the column player. Comparison of GAB and SAB with their baselines.

	A1N	A2N	A3N	Avg.
A1N	-	24.0	5.5	14.8
A2N	76.0	-	27.0	51.5
A3N	94.5	73.0	-	83.8

Table 6.5. Winning rate of the row player against the column player. Comparison of A3N with its baselines.

scheme of GAB and SAB can be effective as both GAS and SAS also search in asymmetrically-abstracted action spaces—the algorithms differ only in their search scheme. Finally, the superiority of A3N against A2N demonstrates the effectiveness of generating asymmetric action abstractions by having a set of unrestricted units for which all unit-actions are considered during search. A2N’s asymmetry relies on the strategies encoded in scripts as all units are restricted to a set of scripts. By contrast, A3N’s asymmetry allows the search procedure to discover strategies different than those encoded in scripts by considering all unit-actions for a small set of units.

6.5 Comparison with State-of-the-Art Algorithms

In this section we evaluate GAB, SAB, A3N against the current state-of-the-art search-based methods for RTS games. Namely, we test the following algorithms: Portfolio Greedy Search (PGS) [1], Stratified Strategy Selection (SSS) [2], Adversarial Hierarchical Task Network (AHT) [36], an algorithm that uses Monte Carlo tree search and HTN planning; NaïveMCTS [4] (henceforth referred as NS); the

	HR	RAR	AHT	NS	WR	A1N	SSS	PGS	PS	SCV	LR	STT	SAB	GAB	A3N	Avg.
HR	-	85.0	13.5	57.5	15.0	17.5	30.0	23.0	16.0	18.0	12.5	5.0	9.0	8.0	3.0	22.4
RAR	15.0	-	57.0	78.0	60.0	24.0	16.0	11.0	3.0	20.0	0.0	16.0	11.0	12.0	1.0	23.1
AHT	86.5	43.0	-	18.5	10.0	13.0	27.0	24.5	39.5	28.5	39.5	9.0	19.5	19.5	2.0	27.1
NS	42.5	22.0	81.5	-	41.0	35.0	22.0	20.0	28.0	26.5	20.0	20.0	27.0	23.0	6.5	29.6
WR	85.0	40.0	90.0	59.0	-	29.0	49.0	43.0	40.0	50.0	35.0	38.5	27.5	31.5	26.5	46.0
A1N	82.5	76.0	87.0	65.0	71.0	-	36.5	34.0	49.0	46.0	34.0	28.0	24.0	26.0	11.5	47.9
SSS	70.0	84.0	73.0	78.0	51.0	63.5	-	54.0	42.0	37.0	28.0	27.0	32.0	17.0	16.0	48.0
PGS	77.0	89.0	75.5	80.0	57.0	66.0	46.0	-	45.5	42.5	46.0	27.5	32.0	13.0	18.5	51.1
PS	84.0	97.0	60.5	72.0	60.0	51.0	58.0	54.5	-	42.5	47.0	30.0	29.0	26.0	28.5	52.9
SCV	82.0	80.0	71.5	73.5	50.0	54.0	63.0	57.5	57.5	-	61.0	40.5	44.0	34.5	23.5	56.6
LR	87.5	100.0	60.5	80.0	65.0	66.0	72.0	54.0	53.0	39.0	-	53.0	35.5	17.0	41.0	58.8
STT	95.0	84.0	91.0	80.0	61.5	72.0	73.0	72.5	70.0	59.5	47.0	-	53.0	29.0	52.5	67.1
SAB	91.0	89.0	80.5	73.0	72.5	76.0	68.0	68.0	71.0	56.0	64.5	47.0	-	36.5	23.0	65.4
GAB	92.0	88.0	80.5	77.0	68.5	74.0	83.0	87.0	74.0	65.5	83.0	71.0	63.5	-	47.5	75.3
A3N	97.0	99.0	98.0	93.5	73.5	88.5	84.0	81.5	71.5	76.5	59.0	47.5	77.0	52.5	-	78.5

Table 6.6. Comparison of GAB, SAB, and A3N with current state-of-the-art search-based methods.

MCTS version of Puppet Search (PS) [14], Strategy Tactics (STT) [15], Strategy Creation via Voting (SCV) [16], and four hard-coded scripts focused in rush, called Light rush (LR), Ranged rush (RR), Heavy rush (HR), and Worker Rush (WR) [35].

Table 6.6 shows the winning rate of the row player against the column player. Overall, A3N wins more matches than any approach tested, suggesting that if a large diversity of maps and opponents are considered, then A3N is the current state-of-the-art in μ RTS. GAB is a close second place with an average winning rate of 75.3. In terms of direct confronts A3N obtains a winning lower than 50.0 only against STT; GAB obtains a winning lower than 50.0 only against A3N. SAB and STT obtain similar average winning rate, 65.4 and 67.1, respectively. Both A3N and STT are able to defeat the weaker opponents by a large margin (e.g., both win almost all matches against the script HR), but A3N is able to better exploit stronger opponents such as PGS, SSS, SAB, and GAB. A3N achieves a larger overall winning rate than STT because of the results against these opponents.

The size of the search space of μ RTS matches is mainly defined by the structure and the size of the map in which the matches take place. Matches played in smaller maps tend to be quicker, with fewer units being controlled by the players at any moment of the match. Matches played in larger maps tend to take longer and the players control a larger number of units, thus increasing size of the search space. The distinction between small and large maps is important because algorithms searching in un-abstracted spaces tend to perform better in smaller maps than algorithms that search in uniformly-abstracted spaces. This is because the search space is small

Small Maps																
-	RAR	HR	PS	SSS	PGS	LR	AHT	SCV	NS	A1N	STT	WR	GAB	SAB	A3N	Avg.
RAR	-	25.0	7.5	25.0	17.5	0.0	22.5	25.0	47.5	15.0	10.0	25.0	30.0	27.5	2.5	20.0
HR	75.0	-	30.0	35.0	32.5	6.3	2.5	20.0	40.0	17.5	5.0	25.0	17.5	22.5	2.5	23.7
PS	92.5	70.0	-	47.5	36.3	45.0	28.8	22.5	30.0	15.0	12.5	12.5	7.5	10.0	16.3	31.9
SSS	75.0	65.0	52.5	-	55.0	50.0	45.0	25.0	47.5	37.5	32.5	27.5	27.5	35.0	17.5	42.3
PGS	82.5	67.5	63.8	45.0	-	47.5	47.5	30.0	50.0	32.5	26.3	25.0	27.5	40.0	15.0	42.9
LR	100.0	93.8	55.0	50.0	52.5	-	26.3	25.0	50.0	37.5	30.0	25.0	25.0	25.0	42.5	45.5
AHT	77.5	97.5	71.3	55.0	52.5	73.8	-	51.3	2.5	32.5	22.5	17.5	43.8	42.5	5.0	46.1
SCV	75.0	80.0	77.5	75.0	70.0	75.0	48.8	-	42.5	40.0	40.0	0.0	40.0	27.5	20.0	50.8
NS	52.5	60.0	70.0	52.5	50.0	50.0	97.5	57.5	-	55.0	47.5	57.5	57.5	67.5	15.0	56.4
A1N	85.0	82.5	85.0	62.5	67.5	62.5	67.5	60.0	45.0	-	50.0	45.0	57.5	55.0	18.8	60.3
STT	90.0	95.0	87.5	67.5	73.8	70.0	77.5	60.0	52.5	50.0	-	36.3	57.5	57.5	32.5	64.8
WR	75.0	75.0	87.5	72.5	75.0	75.0	82.5	100.0	42.5	55.0	63.8	-	73.8	58.8	46.3	70.2
GAB	70.0	82.5	92.5	72.5	72.5	75.0	56.3	60.0	42.5	42.5	42.5	26.3	-	47.5	23.8	57.6
SAB	72.5	77.5	90.0	65.0	60.0	75.0	57.5	72.5	32.5	45.0	42.5	41.3	52.5	-	30.0	58.1
A3N	97.5	97.5	83.8	82.5	85.0	57.5	95.0	80.0	85.0	81.3	67.5	53.8	76.3	70.0	-	79.5
Large Maps																
-	NS	AHT	HR	RAR	WR	A1N	SSS	PGS	SCV	PS	LR	STT	SAB	A3N	GAB	Avg.
NS	-	70.8	30.8	1.7	30.0	21.7	1.7	0.0	5.8	0.0	0.0	1.7	0.0	0.8	0.0	11.8
AHT	29.2	-	79.2	20.0	5.0	0.0	8.3	5.8	13.3	18.3	16.7	0.0	4.2	0.0	3.3	14.5
HR	69.2	20.8	-	91.7	8.3	17.5	26.7	16.7	16.7	6.7	16.7	5.0	0.0	3.3	1.7	21.5
RAR	98.3	80.0	8.3	-	83.3	30.0	10.0	6.7	16.7	0.0	0.0	20.0	0.0	0.0	0.0	25.2
WR	70.0	95.0	91.7	16.7	-	11.7	33.3	21.7	16.7	8.3	8.3	21.7	6.7	13.3	3.3	29.9
A1N	78.3	100.0	82.5	70.0	88.3	-	19.2	11.7	36.7	25.0	15.0	13.3	3.3	6.7	5.0	39.6
SSS	98.3	91.7	73.3	90.0	66.7	80.8	-	53.3	45.0	35.0	13.3	23.3	30.0	15.0	10.0	51.8
PGS	100.0	94.2	83.3	93.3	78.3	88.3	46.7	-	50.8	33.3	45.0	28.3	26.7	20.8	3.3	56.6
SCV	94.2	86.7	83.3	83.3	83.3	63.3	55.0	49.2	-	44.2	51.7	40.8	55.0	25.8	30.8	60.5
PS	100.0	81.7	93.3	100.0	91.7	75.0	65.0	66.7	55.8	-	48.3	41.7	41.7	36.7	38.3	66.8
LR	100.0	83.3	83.3	100.0	91.7	85.0	86.7	55.0	48.3	51.7	-	68.3	42.5	40.0	11.7	67.7
STT	98.3	100.0	95.0	80.0	78.3	86.7	76.7	71.7	59.2	58.3	31.7	-	50.0	65.8	10.0	68.7
SAB	100.0	95.8	100.0	100.0	93.3	96.7	70.0	73.3	45.0	58.3	57.5	50.0	-	18.3	25.8	70.3
A3N	99.2	100.0	96.7	100.0	86.7	93.3	85.0	79.2	74.2	63.3	60.0	34.2	81.7	-	36.7	77.9
GAB	100.0	96.7	98.3	100.0	96.7	95.0	90.0	96.7	69.2	61.7	88.3	90.0	74.2	63.3	-	87.1

Table 6.7. Winning rate of the row player against the column player, divided into small maps (8×8 and 16×6) and large maps (24×24 , 32×32 and 64×64).

enough for the algorithms to encounter strong strategies while accounting for all legal actions. However, the strategy of searching in un-abstracted spaces does not scale to large maps, where algorithms that search in uniformly-abstracted spaces tend to perform better due to their search being focused on the set of promising actions returned by scripts. We hypothesize that asymmetric action abstractions allow search algorithms to derive strong strategies in both small and large maps.

Table 6.7 shows the winning rate of search algorithms in small maps (top) and large maps (bottom). The small maps consist of the two maps of size 8×8 and the two of size 16×16 . The large maps consist of the two maps of size 24×24 , two of size 32×32 , and two of size 64×64 . We observe that NS, an algorithm that searches in un-abstracted spaces, outperforms algorithms such as PGS and SSS on average on the small maps. While NS has an average winning rate of 56.4, PGS and SSS have

average winning rates of 42.9 and 42.3, respectively. However, on large maps, NS's average winning rate is only 11.8, while PGS and SSS's average winning rate is of 56.6 and 51.8, respectively. The results shown in Table 6.7 support our hypothesis that algorithms searching in asymmetrically-abstracted spaces can perform well in both small and large maps as GAB, SAB, and A3N perform well in both settings. In particular, A3N performs consistently well in the two scenarios. GAB defeats all other search algorithms in the large maps, obtaining a 87.1 average winning rate. While its performance on the smaller maps still indicates superiority over other approaches, winning rate of 57.6, the results are weaker than in the large maps.

These results suggest that the performance of algorithms searching in asymmetric action abstractions can be further improved if the asymmetric action abstraction was map-dependent, i.e., the size of the set of unrestricted units N and the strategies for selecting such a set was dependent on the map. In our experiments we use one strategy and one value of N for all maps. While the setting used for GAB is good for large maps, it might be hurting the performance of the algorithm on small maps, where the value of N could likely be larger. An interesting direction for future research is to learn policies for selecting the unrestricted units based on the maps being used.

Chapter 7

Conclusions

In this dissertation we introduced asymmetric action abstractions for multi-unit zero-sum extensive-form games. We also introduced A2N, A3N, GAB, and SAB, four search algorithms for searching in asymmetrically-abstracted spaces. Similarly to uniformly-abstracted spaces, asymmetric abstractions also use domain-knowledge in the form of scripts. However, in contrast with uniform abstractions, which restrict all units to the unit-actions returned by the scripts, asymmetric action abstractions restrict only a subset of the units—the restricted units. Algorithms searching with asymmetric action abstractions account for all legal unit-actions of the remaining units—the unrestricted units. As a result, the strategy derived by search algorithms are focused on the unrestricted units, as the algorithms are able to derive finer plans for such units. Asymmetric action abstractions can be seen as an attention scheme, where the search “pays more attention” to a subset of units.

We evaluated our algorithms with an extensive set of experiments on μ RTS. Our results suggest that A3N is the current state-of-the-art algorithm in this domain if one considers a large diversity of maps and opponents, similarly to the setting used in the μ RTS annual competition [5]. If one considers large maps such as those used in commercial games, then GAB presented the strongest results. Although we performed our experiments on μ RTS, the ideas of this dissertation are general and could be applied to other games. For example, in collectible card games such as Hearthstone [37] and Magic: The Gathering [38] the player has to decide on the action of several cards. Algorithms could use asymmetric action abstractions to focus their search on a subset of the cards. The ideas introduced in this dissertation might also be applied in problems other than games. For example, a robotic system that controls several actuators while trying to accomplish a task can benefit from asymmetric abstractions. This is because some actuators might require a finer

control than the others.

Bibliography

- [1] D. Churchill and M. Buro, “Portfolio greedy search and simulation for large-scale combat in StarCraft.,” in *Proceedings of the Conference on Computational Intelligence in Games*, pp. 1–8, IEEE, 2013.
- [2] L. H. S. Lelis, “Stratified strategy selection for unit control in real-time strategy games,” in *International Joint Conference on Artificial Intelligence*, pp. 3735–3741, 2017.
- [3] S. Ontañón, “The combinatorial multi-armed bandit problem and its application to real-time strategy games,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 58–64, 2013.
- [4] S. Ontañón, “Combinatorial multi-armed bandits for real-time strategy games,” *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, 2017.
- [5] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, and L. H. Lelis, “The first microrsts artificial intelligence competition.,” *AI Magazine*, vol. 39, no. 1, 2018.
- [6] R. O. Moraes and L. H. S. Lelis, “Asymmetric action abstractions for multi-unit control in adversarial real-time scenarios,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 876–883, AAAI, 2018.
- [7] R. O. Moraes, J. R. H. Mariño, L. H. S. Lelis, and M. A. Nascimento, “Action abstractions for combinatorial multi-armed bandit tree search,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 74–80, AAAI, 2018.
- [8] M. Chung, M. Buro, and J. Schaeffer, “Monte Carlo planning in RTS games,” in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005.

- [9] F. Sailer, M. Buro, and M. Lanctot, “Adversarial planning through strategy simulation,” in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 80–87, 2007.
- [10] R.-K. Balla and A. Fern, “UCT for tactical assault planning in real-time strategy games,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 40–45, 2009.
- [11] D. Churchill, A. Saffidine, and M. Buro, “Fast heuristic search for RTS game combat scenarios,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.
- [12] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *Proceedings of the European Conference on Machine Learning*, pp. 282–293, Springer-Verlag, 2006.
- [13] D. Churchill and M. Buro, “Hierarchical portfolio search: Prismata’s robust AI architecture for games with large search spaces,” in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 16–22, 2015.
- [14] N. A. Barriga, M. Stanescu, and M. Buro, “Game tree search based on non-deterministic action scripts in real-time strategy games,” *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
- [15] N. A. Barriga, M. Stanescu, and M. Buro, “Combining strategic learning and tactical search in real-time strategy games,” *The AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.
- [16] C. R. Silva, R. O. Moraes, L. H. S. Lelis, and Y. Gal, “Strategy generation for multi-unit real-time games via voting,” *IEEE Transactions on Games*, 2018.
- [17] A. Shleyfman, A. Komenda, and C. Domshlak, “On combinatorial actions and cmabs with linear side information,” in *European Conference on Artificial Intelligence*, pp. 825–830, 2014.
- [18] G.-J. Roelofs, “Action Space Representation in Combinatorial Multi-Armed Bandits,” Master’s thesis, Maastricht University, The Netherlands, 2015.
- [19] O. Vinyals *et al.*, “AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.” <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>, 2019.

- [20] A. Gilpin and T. Sandholm, “Better automated abstraction techniques for imperfect information games, with application to texas hold’em poker,” in *International Joint Conference on Autonomous Agents and Multiagent Systems*, (New York, NY, USA), pp. 192:1–192:8, ACM, 2007.
- [21] J. A. Hawkin, R. Holte, and D. Szafron, “Automated action abstraction of imperfect information extensive-form games,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 681–687, 2011.
- [22] M. Johanson, N. Burch, R. Valenzano, and M. Bowling, “Evaluating state-space abstractions in extensive-form games,” in *International Conference on Autonomous Agents and Multi-agent Systems*, pp. 271–278, International Foundation for Autonomous Agents and Multiagent Systems, 2013.
- [23] N. Bard, M. Johanson, and M. Bowling, “Asymmetric abstractions for adversarial settings,” in *International Conference on Autonomous Agents and Multi-agent Systems*, pp. 501–508, International Foundation for Autonomous Agents and Multiagent Systems, 2014.
- [24] B. Bosanský, V. Lisý, M. Lancot, J. Cermák, and M. H. M. Winands, “Algorithms for computing strategies in two-player simultaneous move games,” *Artificial Intelligence*, vol. 237, pp. 1–40, 2016.
- [25] D. E. Knuth and R. W. Moore, “An analysis of alpha-beta pruning,” *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
- [26] S. J. Russell and P. Norvig, “Artificial intelligence: a modern approach (international edition),” 2002.
- [27] L. Atkin and D. Slate, “Computer chess compendium,” ch. Chess 4.5-The Northwestern University Chess Program, pp. 80–103, Berlin, Heidelberg: Springer-Verlag, 1988.
- [28] A. L. Zobrist, “A new hashing method with application for game playing,” 1990.
- [29] R. O. Moraes, J. R. H. Mariño, and L. H. S. Lelis, “Nested-greedy search for adversarial real-time games,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 67–73, 2018.
- [30] R. E. Korf, “Macro-operators: A weak method for learning,” *Artificial Intelligence*, vol. 26, no. 1, pp. 35–77, 1985.

- [31] J. E. Laird, P. S. Rosenbloom, and A. Newell, “Chunking in soar: The anatomy of a general learning mechanism,” *Machine Learning*, vol. 1, no. 1, pp. 11–46, 1986.
- [32] G. A. Iba, “A heuristic approach to the discovery of macro-operators,” *Machine Learning*, vol. 3, no. 4, pp. 285–317, 1989.
- [33] R. S. Sutton, D. Precup, and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [34] M. C. Machado, M. G. Bellemare, and M. Bowling, “A laplacian framework for option discovery in reinforcement learning,” in *Proceedings of the International Conference on Machine Learning*, pp. 2295–2304, 2017.
- [35] M. Stanescu, N. A. Barriga, A. Hess, and M. Buro, “Evaluating real-time strategy game states using convolutional neural networks,” in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pp. 1–7, IEEE, 2016.
- [36] S. Ontañón and M. Buro, “Adversarial hierarchical-task network planning for complex real-time games,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1658, 2015.
- [37] A. Dockhorn and S. Mostaghim, “Introducing the hearthstone-ai competition,” 2019.
- [38] C. D. Ward and P. I. Cowling, “Monte carlo search applied to card selection in magic: The gathering,” in *Proceedings of the International Conference on Computational Intelligence and Games*, pp. 9–16, IEEE Press, 2009.

Appendix A

Proofs

The proof of Theorem 1 hinges on the fact that one has access to more actions with Ω than with Φ . This idea is formalized in Lemma 1.

Lemma 1 *Let Φ be a uniform abstraction and Ω be an asymmetric abstraction, both defined with the same set of scripts \mathcal{P} . Also, let $\mathcal{A}'_i(s)$ be the set of actions available at state s according to Φ and $\mathcal{A}''_i(s)$ the set of actions available at s according to Ω . $\mathcal{A}'_i(s) \subseteq \mathcal{A}''_i(s)$ for any s .*

Proof. By definition, the actions in $\mathcal{A}'_i(s)$ are generated by the Cartesian product of $\mathcal{M}(s, u, \mathcal{P})$ for all u in \mathcal{U}_i in s . The actions in $\mathcal{A}''_i(s)$ are generated by the Cartesian product of $\mathcal{M}(s, u, \mathcal{P})$ for all u in $\mathcal{U}_i \setminus \mathcal{U}'_i$ and of $\mathcal{M}(s, u)$ for all u in \mathcal{U}'_i . Since, also by definition, $\mathcal{M}(s, u, \mathcal{P}) \subseteq \mathcal{M}(s, u)$, we have that $\mathcal{A}'_i(s) \subseteq \mathcal{A}''_i(s)$. \square

Let Σ'_i and Σ''_i be the set of player i 's strategies whose supports contain only actions in \mathcal{A}'_i and \mathcal{A}''_i , respectively. Also, let Σ_{-i} be the set of all player $-i$'s strategies. Lemma 1 allows us to write the following corollary.

Corollary 1 *For abstractions Φ and Ω defined from the same set of scripts \mathcal{P} we have that $\Sigma'_i \subseteq \Sigma''_i$.*

Theorem 1 *Let Φ be a uniform abstraction and Ω be an asymmetric abstraction, both defined with the same set of scripts \mathcal{P} . For a zero-sum extensive-form game ∇ with start state s , let $V_i^\Phi(s)$ be the optimal value of the game computed by considering the game tree induced by Φ ; define $V_i^\Omega(s)$ analogously. We have that $V_i^\Omega(s) \geq V_i^\Phi(s)$.*

Proof. We prove the theorem by induction on the level of the game tree. The base case is given by leaf nodes s_l . Since $V_i^\Omega(s_l) = V_i^\Phi(s_l) = \mathcal{R}_i(s_l)$, the theorem holds.

The inductive hypothesis is that $V_i^\Omega(s') \geq V_i^\Phi(s')$ for any state s' at level $j + 1$ of the tree. For any state s at level j we have that,

$$\begin{aligned}
 V_i^\Omega(s) &= \max_{\sigma_i \in \Sigma_i''} \min_{\sigma_{-i} \in \Sigma_{-i}} \sum_{a_i \in \mathcal{A}(s)} \sum_{a_{-i} \in \mathcal{A}_{-i}(s)} \sigma_i(s, a_i) \cdot \\
 &\quad \sigma_{-i}(s, a_{-i}) \cdot V_i^\Omega(\mathcal{T}(s, a_i, a_{-i})) \\
 &\geq \max_{\sigma_i \in \Sigma_i'} \min_{\sigma_{-i} \in \Sigma_{-i}} \sum_{a_i \in \mathcal{A}(s)} \sum_{a_{-i} \in \mathcal{A}_{-i}(s)} \sigma_i(s, a_i) \cdot \\
 &\quad \sigma_{-i}(s, a_{-i}) \cdot V_i^\Phi(\mathcal{T}(s, a_i, a_{-i})) = V_i^\Phi(s).
 \end{aligned}$$

The first equality is the definition of the value of a zero-sum simultaneous move game. The inequality is because $\Sigma_i' \subseteq \Sigma_i''$ (Corollary 1) and $V_i^\Omega(\mathcal{T}(s, a_i, a_{-i})) \geq V_i^\Phi(\mathcal{T}(s, a_i, a_{-i}))$, as $\mathcal{T}(s, a_i, a_{-i})$ returns a state at level $j + 1$ of the tree (inductive hypothesis). The inequality also holds if the transition $\mathcal{T}(s, a_i, a_{-i})$ returns a terminal state z at level $j + 1$ as $V_i^\Omega(z) = V_i^\Phi(z) = \mathcal{R}_i(z)$. The last equality is analogous to the first one. \square

Appendix B

Evaluating Strategies and Number of Unrestricted Units for Each Map to GAB

This appendix shows, in details, each one of the matches compiled at section 6.3 to define the best setting for GAB.

GAB vs. PGS - Map basesWorkers8x8A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	8.00	9.00	9.00	8.00	5.00	8.00	6.00	3.00	6.00	5.50	6.75
FC	9.00	9.00	9.00	6.00	1.00	4.00	8.00	6.00	8.00	8.00	6.80
CE	9.00	7.00	4.00	6.00	6.50	5.00	3.00	7.00	5.00	4.00	5.65
FE	10.00	9.00	8.00	9.00	9.00	5.00	10.00	6.00	7.00	5.00	7.80
AV-	10.00	9.00	8.00	8.00	4.00	8.00	4.00	7.00	4.00	7.00	6.90
AV+	10.00	10.00	8.00	6.00	4.00	3.00	5.50	5.50	4.00	4.00	6.00
HP-	10.00	9.00	10.00	7.00	5.00	7.00	7.00	7.00	8.00	7.00	7.70
HP+	8.00	7.00	6.00	5.00	8.00	3.00	7.00	7.00	4.00	6.00	6.10
R	10.00	6.00	9.00	7.00	6.00	3.00	5.00	5.00	8.00	8.00	6.70

Table B.1. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map FourBasesWorkers8x8											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	10.00	10.00	10.00	10.00	10.00	10.00	9.00	6.00	9.00	9.40
FC	10.00	10.00	9.00	10.00	10.00	10.00	8.00	9.00	9.00	6.50	9.15
CE	9.50	10.00	10.00	7.50	7.00	6.50	6.00	4.50	6.00	8.00	7.50
FE	10.00	10.00	10.00	10.00	10.00	10.00	10.00	9.00	10.00	8.00	9.70
AV-	10.00	10.00	10.00	10.00	9.00	9.00	10.00	8.00	8.00	6.00	9.00
AV+	10.00	10.00	10.00	10.00	9.00	10.00	10.00	8.00	6.00	7.00	9.00
HP-	10.00	10.00	10.00	9.00	9.00	10.00	10.00	8.00	8.00	6.50	9.05
HP+	9.00	10.00	10.00	9.00	9.00	9.00	7.00	6.00	8.00	5.00	8.20
R	10.00	10.00	10.00	9.00	10.00	10.00	9.00	9.00	10.00	9.00	9.60

Table B.2. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map basesWorkers16x16A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	3.00	5.00	3.00	3.00	0.00	4.00	5.00	3.00	5.00	4.00	3.50
FC	3.00	4.00	3.00	2.00	1.00	1.00	2.00	4.00	5.00	1.00	2.60
CE	2.00	0.00	0.00	0.00	1.00	0.00	1.00	2.00	1.00	3.00	1.00
FE	8.00	7.00	2.00	6.00	5.00	5.00	5.00	7.00	4.00	4.00	5.30
AV-	5.00	3.00	4.00	1.00	2.00	1.00	3.00	2.00	0.00	1.00	2.20
AV+	6.00	3.00	4.00	0.00	0.00	3.00	2.00	0.00	5.00	6.00	2.90
HP-	6.00	4.00	4.00	3.00	1.00	1.00	1.00	1.00	1.00	4.00	2.60
HP+	3.00	2.00	6.00	4.00	4.00	4.00	3.00	4.00	3.00	2.00	3.50
R	3.00	3.00	2.00	0.00	3.00	3.00	2.00	4.00	6.00	3.00	2.90

Table B.3. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map TwoBasesBarracks16x16											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	1.00	2.00	0.00	4.00	1.00	4.00	1.00	7.00	3.00	5.00	2.80
FC	3.00	3.00	1.00	3.00	4.00	6.00	6.00	3.00	5.00	4.00	3.80
CE	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00	4.00	7.00	1.30
FE	1.00	0.00	0.00	2.00	2.00	2.00	4.00	1.00	5.00	5.00	2.20
AV-	4.00	3.00	3.00	2.00	2.00	1.00	2.00	3.00	1.00	1.00	2.20
AV+	2.00	1.00	0.00	1.00	2.00	4.00	2.00	2.00	0.00	6.00	2.00
HP-	2.00	0.00	2.00	1.00	0.00	1.00	1.00	2.00	2.00	0.00	1.10
HP+	0.00	4.00	2.00	3.00	3.00	2.00	6.00	3.00	5.00	3.00	3.10
R	0.00	0.00	2.00	1.00	3.00	7.00	5.00	2.00	8.00	6.00	3.40

Table B.4. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map basesWorkers24x24A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	5.00	5.50	3.00	6.00	6.00	3.00	6.00	2.00	6.00	3.00	4.55
FC	7.00	6.00	5.00	3.00	5.00	3.00	3.00	2.00	4.00	3.00	4.10
CE	9.00	8.00	8.00	7.00	6.00	5.00	8.00	3.00	4.50	5.50	6.40
FE	9.00	9.00	7.00	9.00	8.00	4.00	4.00	5.00	4.50	0.00	5.95
AV-	8.00	8.00	10.00	9.00	8.00	8.00	9.00	6.00	6.00	6.00	7.80
AV+	6.00	9.00	10.00	8.00	8.00	10.00	9.00	6.00	6.00	5.00	7.70
HP-	6.00	8.00	8.00	9.00	9.00	8.00	8.00	6.00	8.00	7.00	7.70
HP+	4.00	7.00	2.00	1.00	4.00	1.00	3.00	2.00	2.00	3.00	2.90
R	8.00	3.50	5.00	7.00	4.00	2.00	4.00	6.50	5.00	4.00	4.90

Table B.5. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map basesWorkers24x24A Barrack											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	6.00	5.00	2.00	3.00	5.00	3.00	5.00	3.00	4.00	5.00	4.10
FC	6.00	3.00	2.50	3.00	4.00	2.00	1.00	2.50	5.00	3.00	3.20
CE	8.00	8.00	7.00	7.00	8.00	6.00	8.00	5.00	3.00	4.00	6.40
FE	8.00	10.00	9.00	9.00	9.00	6.00	5.00	1.00	4.00	5.00	6.60
AV-	9.00	8.00	10.00	10.00	9.00	8.00	6.00	6.00	6.00	5.00	7.70
AV+	7.00	9.00	10.00	9.00	9.00	9.00	8.00	6.00	4.00	8.00	7.90
HP-	7.00	10.00	10.00	8.00	9.00	9.00	4.00	8.00	5.00	3.00	7.30
HP+	7.50	7.00	6.50	4.00	2.00	0.00	4.00	2.00	0.00	5.00	3.80
R	8.00	3.00	2.00	3.00	3.00	1.00	5.00	5.00	5.00	6.00	4.10

Table B.6. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map basesWorkers32x32A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	4.00	6.00	7.00	8.00	4.00	4.00	2.00	4.00	2.00	3.00	4.40
FC	8.00	6.00	6.00	10.00	3.00	2.00	3.00	3.00	3.00	4.00	4.80
CE	8.00	10.00	8.00	10.00	6.00	7.00	4.00	5.00	9.00	5.00	7.20
FE	10.00	10.00	7.00	6.00	4.00	4.00	4.00	3.50	3.00	2.50	5.40
AV-	9.00	9.00	9.00	10.00	10.00	9.00	7.00	8.00	7.00	8.00	8.60
AV+	9.00	8.00	8.00	10.00	9.00	9.00	10.00	8.00	9.00	9.00	8.90
HP-	9.00	9.00	9.00	9.00	8.00	10.00	10.00	7.00	10.00	9.00	9.00
HP+	9.00	5.00	6.00	2.00	3.00	1.00	2.50	0.50	3.00	4.00	3.60
R	7.00	7.00	6.00	6.00	5.00	4.00	3.00	3.00	3.00	2.00	4.60

Table B.7. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map basesWorkersBarracks32x32											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	8.00	5.00	1.00	3.00	3.00	2.00	3.00	4.00	4.00	2.00	3.50
FC	6.00	2.00	1.00	5.00	5.00	5.00	4.00	5.00	5.00	4.50	4.25
CE	8.00	9.00	9.00	7.00	8.00	4.00	5.00	4.00	6.00	8.00	6.80
FE	10.00	8.00	10.00	10.00	10.00	6.00	4.50	3.00	1.00	4.00	6.65
AV-	10.00	9.00	9.00	10.00	10.00	9.00	9.00	9.00	8.00	5.00	8.80
AV+	10.00	8.00	9.00	10.00	10.00	10.00	9.00	6.00	9.00	7.00	8.80
HP-	9.00	9.00	10.00	10.00	7.00	8.00	8.00	8.00	8.00	8.00	8.50
HP+	9.00	8.00	8.00	5.00	2.00	5.00	4.00	5.00	3.00	5.00	5.40
R	7.00	3.00	2.00	4.00	3.00	3.00	5.00	3.00	3.00	1.00	3.40

Table B.8. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map BloodBath.scmB											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	7.00	5.00	5.00	6.00	5.00	4.00	3.00	6.00	2.00	5.00	4.80
FC	6.00	2.00	1.00	5.00	5.00	5.00	4.00	5.00	5.00	4.50	4.25
CE	8.00	7.00	9.00	7.00	8.00	9.00	7.00	8.00	4.00	7.00	7.40
FE	7.00	6.50	7.00	5.50	5.00	6.50	2.00	1.00	6.00	5.00	5.15
AV-	6.00	8.00	10.00	10.00	9.00	8.00	8.00	9.00	6.00	6.00	8.00
AV+	4.00	7.00	9.00	10.00	10.00	9.00	10.00	9.00	7.00	8.00	8.30
HP-	5.00	9.00	10.00	10.00	8.00	10.00	8.50	8.00	9.00	7.00	8.45
HP+	7.00	5.00	6.00	2.50	4.00	5.00	5.00	2.00	2.00	0.00	3.85
R	3.00	8.00	4.00	5.00	3.00	4.50	6.00	3.00	2.50	4.00	4.30

Table B.9. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

GAB vs. PGS - Map BloodBath.scmD											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	7.00	7.00	7.00	6.00	4.00	4.00	2.00	3.00	7.50	4.00	5.15
FC	7.00	5.00	6.00	1.00	4.00	4.00	4.00	2.00	4.00	2.00	3.90
CE	6.00	9.00	7.00	9.00	8.50	9.50	7.00	5.00	6.50	5.00	7.25
FE	3.00	5.50	9.00	8.00	5.00	6.00	4.00	2.00	2.00	3.00	4.75
AV-	6.00	7.00	9.00	9.00	10.00	9.00	10.00	6.00	7.00	9.00	8.20
AV+	5.00	6.00	8.00	10.00	9.00	10.00	9.00	7.00	5.50	7.00	7.65
HP-	4.00	4.00	9.00	10.00	10.00	9.00	8.00	8.00	8.00	9.50	7.95
HP+	5.00	2.00	4.50	4.00	4.00	6.00	6.00	4.00	8.00	3.00	4.65
R	7.50	4.00	2.50	3.00	7.50	5.00	5.00	5.00	6.00	5.50	5.10

Table B.10. Winning rate of variants of GAB against PGS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

Appendix C

Evaluating Strategies and Number of Unrestricted Units for Each Map to SAB

This appendix shows, in details, each one of the matches compiled at section 6.3 to define the best setting for SAB.

SAB vs. SSS - Map basesWorkers8x8A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	10.00	8.00	8.00	5.00	7.00	5.00	8.00	7.00	8.00	7.00
FC	8.00	7.50	9.00	6.00	9.00	6.00	7.00	3.00	4.00	4.00	5.86
CE	9.00	7.00	3.00	5.00	7.00	5.50	7.00	3.50	4.00	5.00	5.45
FE	10.00	10.00	9.00	9.00	8.00	6.00	4.00	4.00	7.00	6.00	6.82
AV-	10.00	9.00	8.00	9.00	4.00	6.00	4.00	4.00	5.00	5.00	6.00
AV+	10.00	10.00	10.00	7.00	8.00	7.00	6.50	5.00	6.00	5.00	7.14
HP-	10.00	10.00	10.00	6.00	4.00	7.00	3.00	8.00	3.00	7.00	6.45
HP+	10.00	10.00	6.00	7.00	6.00	7.00	4.00	8.00	6.00	8.00	6.91
R	10.00	9.00	8.00	8.00	8.00	6.00	8.00	6.00	7.00	6.00	7.05

Table C.1. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map FourBasesWorkers8x8											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	10.00	10.00	10.00	9.00	10.00	9.00	8.00	9.00	8.00	9.30
FC	10.00	10.00	10.00	10.00	9.00	10.00	9.00	9.00	9.00	5.00	9.10
CE	10.00	10.00	10.00	8.50	6.00	7.00	6.00	8.00	7.00	4.00	7.65
FE	9.00	10.00	10.00	10.00	10.00	9.00	10.00	9.00	10.00	9.00	9.60
AV-	10.00	10.00	10.00	10.00	9.00	8.00	9.00	9.00	8.00	7.00	9.00
AV+	9.00	10.00	10.00	10.00	9.00	9.00	7.00	7.00	8.00	4.00	8.30
HP-	9.00	10.00	10.00	10.00	8.00	7.00	8.00	5.00	7.00	8.00	8.20
HP+	10.00	10.00	10.00	10.00	9.00	8.00	10.00	8.00	9.00	6.00	9.00
R	10.00	10.00	10.00	10.00	10.00	9.00	9.00	8.00	7.00	8.00	9.10

Table C.2. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map TwoBasesBarracks16x16											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	0.00	1.00	2.00	1.00	2.00	3.00	5.00	2.00	3.00	3.00	2.20
FC	1.00	1.00	0.00	2.00	3.00	5.00	4.00	7.00	4.00	6.00	3.30
CE	1.00	0.00	0.00	0.00	0.00	1.00	2.00	1.00	1.00	0.00	0.60
FE	1.00	2.00	2.00	1.00	1.00	5.00	8.00	5.00	5.00	5.00	3.50
AV-	1.00	1.00	2.00	2.00	2.00	3.00	2.00	1.00	1.00	1.00	1.60
AV+	1.00	1.00	1.00	2.00	2.00	1.00	2.00	3.00	0.00	4.00	1.70
HP-	1.00	1.00	1.00	2.00	1.00	2.00	1.00	2.00	2.00	4.00	1.70
HP+	1.00	0.00	0.00	1.00	2.00	0.00	5.00	5.00	5.00	5.00	2.40
R	1.00	0.00	1.00	3.00	6.00	3.00	5.00	5.00	5.00	5.00	3.40

Table C.3. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map basesWorkers16x16A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	6.00	5.00	2.00	5.00	2.00	4.00	3.00	5.00	6.00	3.00	4.10
FC	5.00	6.00	2.00	5.00	3.00	1.00	2.00	5.00	6.00	7.00	4.20
CE	1.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	2.00	0.00	0.50
FE	7.00	8.00	5.00	6.00	4.00	3.00	9.00	6.00	3.00	4.00	5.50
AV-	10.00	7.00	2.00	0.00	4.00	2.00	7.00	5.00	3.00	3.00	4.30
AV+	7.00	3.00	5.00	4.00	3.00	2.00	6.00	2.00	6.00	6.00	4.40
HP-	5.00	6.00	4.00	2.00	3.00	4.00	3.00	4.00	4.00	2.00	3.70
HP+	2.00	4.00	6.00	3.00	2.00	6.00	8.00	7.00	7.00	6.00	5.10
R	6.00	6.00	5.00	1.00	3.00	2.00	6.00	6.00	6.00	4.00	4.50

Table C.4. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map basesWorkers24x24A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	7.00	5.00	7.00	3.00	5.00	5.00	4.00	4.50	6.00	3.00	4.95
FC	7.00	6.00	5.00	4.00	6.00	7.00	6.00	2.00	6.00	2.00	5.10
CE	9.00	7.00	5.00	8.00	4.00	6.00	8.00	5.00	7.00	4.00	6.30
FE	7.00	5.00	7.00	5.00	7.00	7.00	3.00	4.00	3.00	4.00	5.20
AV-	6.00	7.00	8.00	9.00	8.00	7.00	9.00	9.00	6.00	7.00	7.60
AV+	5.00	10.00	6.00	7.00	7.00	8.00	7.00	8.00	5.50	7.00	7.05
HP-	6.00	8.00	7.00	5.00	7.00	6.50	6.00	7.00	8.00	7.00	6.75
HP+	3.00	4.00	4.00	2.00	2.00	1.00	4.00	3.00	6.00	4.00	3.30
R	5.00	3.00	2.00	5.00	5.00	4.00	5.00	3.00	4.00	4.00	4.00

Table C.5. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map basesWorkers24x24A Barrack											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	6.00	6.50	3.00	3.00	6.00	3.00	3.00	5.00	3.00	4.00	4.25
FC	3.00	6.00	2.00	3.00	4.00	6.00	3.00	3.00	5.00	8.00	4.30
CE	7.00	4.00	6.00	6.00	4.00	6.00	6.00	8.00	6.00	4.00	5.70
FE	6.00	9.00	8.00	7.00	4.00	7.00	7.00	6.00	3.00	3.00	6.00
AV-	8.00	6.00	8.00	6.00	7.00	7.00	5.00	6.00	2.00	4.00	5.90
AV+	7.00	8.00	6.00	5.00	8.00	8.00	5.00	6.00	3.00	4.00	6.00
HP-	10.00	7.00	7.00	8.00	6.00	4.00	8.00	8.00	7.00	6.00	7.10
HP+	4.00	4.00	6.00	3.00	3.50	4.00	5.00	5.00	3.00	3.00	4.05
R	6.00	4.00	4.00	5.00	3.00	8.00	3.00	7.00	5.00	5.00	5.00

Table C.6. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map basesWorkers32x32A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	7.00	6.00	3.00	6.00	5.00	5.00	5.00	5.00	4.00	7.00	5.30
FC	4.00	7.00	6.00	8.00	5.00	4.00	2.00	4.00	6.00	3.00	4.90
CE	10.00	6.00	5.00	8.00	5.00	9.00	7.00	5.00	8.00	5.00	6.80
FE	4.00	7.00	8.00	9.00	5.00	3.00	4.00	5.00	5.00	6.00	5.60
AV-	7.00	7.00	7.00	5.00	8.00	4.00	6.00	7.00	7.00	6.00	6.40
AV+	6.00	8.00	7.00	8.00	6.00	10.00	8.00	9.00	9.00	7.00	7.80
HP-	7.00	8.00	10.00	8.00	8.00	9.00	7.00	8.00	8.00	8.00	8.10
HP+	4.00	4.00	4.00	5.00	5.00	4.00	1.00	5.00	5.00	3.00	4.00
R	6.00	4.00	9.00	6.00	4.00	5.00	5.00	6.00	4.00	5.00	5.40

Table C.7. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map basesWorkersBarracks32x32											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	7.00	6.00	4.00	7.00	4.00	8.00	6.00	4.00	6.00	6.00	5.80
FC	5.00	6.00	3.00	7.00	4.00	4.00	2.00	6.00	4.00	5.00	4.60
CE	9.00	7.00	7.00	6.00	8.00	5.00	3.00	4.00	6.00	7.00	6.20
FE	5.00	7.00	7.00	5.00	5.00	4.00	3.00	5.00	5.00	4.00	5.00
AV-	7.00	8.00	8.00	9.00	6.00	6.00	8.00	4.00	5.00	7.00	6.80
AV+	6.00	7.00	7.00	8.00	8.00	6.00	6.00	6.00	5.00	6.00	6.50
HP-	9.00	6.00	7.00	6.00	7.00	8.00	5.00	8.00	7.00	9.00	7.20
HP+	6.00	4.00	6.00	4.00	6.00	7.00	5.00	3.00	6.00	6.00	5.30
R	8.00	5.00	5.00	4.00	8.00	3.00	6.00	7.00	5.00	6.00	5.70

Table C.8. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map BloodBath.scmB											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	6.00	9.00	7.00	9.00	6.00	7.00	9.00	8.00	9.00	9.00	7.90
FC	7.00	6.00	8.00	8.00	6.00	7.50	8.00	6.00	8.00	9.00	7.35
CE	8.00	10.00	10.00	10.00	10.00	8.00	8.00	9.00	8.00	9.00	9.00
FE	8.00	8.00	9.00	9.00	8.00	7.00	8.00	8.00	8.00	9.00	8.20
AV-	4.00	7.00	10.00	9.00	10.00	10.00	10.00	10.00	9.00	10.00	8.90
AV+	7.00	9.00	10.00	10.00	10.00	9.00	8.00	10.00	10.00	10.00	9.30
HP-	7.50	8.00	9.00	10.00	10.00	10.00	10.00	10.00	10.00	9.00	9.35
HP+	7.00	8.00	6.00	9.00	8.00	8.00	9.00	7.00	6.00	6.00	7.40
R	7.00	6.00	9.00	10.00	9.00	10.00	7.00	7.00	9.00	8.00	8.20

Table C.9. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

SAB vs. SSS - Map BloodBath.scmD											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	6.00	8.00	8.00	5.00	6.00	8.00	10.00	8.50	8.00	6.00	7.35
FC	6.00	6.00	6.00	8.50	6.00	6.00	8.00	9.00	6.50	7.50	6.95
CE	8.00	9.00	10.00	9.00	10.00	8.50	7.00	7.00	6.00	10.00	8.45
FE	6.00	7.00	10.00	8.00	7.00	9.50	4.00	6.00	8.00	5.50	7.10
AV-	7.00	7.00	10.00	8.00	9.00	10.00	10.00	10.00	8.00	10.00	8.90
AV+	8.00	8.00	10.00	10.00	10.00	8.00	10.00	10.00	9.00	8.50	9.15
HP-	7.00	7.00	7.00	10.00	7.00	10.00	8.00	9.00	10.00	8.00	8.30
HP+	7.00	10.00	9.00	8.50	7.00	5.50	7.00	8.00	7.50	7.50	7.70
R	7.50	7.00	7.00	4.50	6.00	6.00	4.00	7.00	7.50	9.00	6.55

Table C.10. Winning rate of variants of SAB against SSS in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

Appendix D

Evaluating Strategies and Number of Unrestricted Units for Each Map to A3N

This appendix shows, in details, each one of the matches compiled at section 6.3 to define the best setting for A3N.

A3N vs. A1N - Map basesWorkers8x8A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	0.00	0.00	0.00	5.00	10.00	10.00	10.00	10.00	10.00	10.00	6.50
FC	0.00	2.00	2.00	10.00	10.00	10.00	10.00	9.00	10.00	10.00	7.30
CE	1.00	3.00	9.50	9.00	10.00	10.00	10.00	10.00	10.00	10.00	8.25
FE	0.00	0.00	0.00	1.00	5.00	8.50	9.50	10.00	10.00	10.00	5.40
AV-	0.00	0.00	3.00	6.00	9.00	10.00	10.00	10.00	10.00	10.00	6.80
AV+	0.00	1.00	0.00	9.00	9.00	9.00	10.00	10.00	10.00	10.00	6.80
HP-	0.00	0.00	4.00	7.00	9.00	9.00	10.00	10.00	10.00	10.00	6.90
HP+	0.00	2.00	2.00	10.00	9.00	10.00	9.00	10.00	9.00	10.00	7.10
R	0.00	1.00	0.00	5.50	9.00	10.00	10.00	10.00	10.00	10.00	6.55

Table D.1. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map FourBasesWorkers8x8											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	2.00	6.00	1.00
FC	0.00	0.00	0.00	0.00	0.00	0.00	1.00	5.00	8.00	8.00	2.20
CE	0.00	0.00	0.00	2.00	6.00	9.00	10.00	10.00	10.00	10.00	5.70
FE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.10
AV-	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	2.00	0.30
AV+	0.00	0.00	0.00	0.00	1.00	3.00	6.00	7.00	8.00	10.00	3.50
HP-	0.00	0.00	0.00	1.00	0.00	0.50	4.00	9.00	10.00	10.00	3.45
HP+	0.00	0.00	0.00	0.00	0.00	1.00	3.00	5.00	10.00	9.00	2.80
R	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	5.00	9.00	1.50

Table D.2. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map TwoBasesBarracks16x16											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	8.00	8.00	9.00	6.00	7.00	8.00	5.00	6.00	3.00	4.00	6.40
FC	9.00	10.00	10.00	10.00	8.00	10.00	9.00	8.00	5.00	6.00	8.50
CE	10.00	9.00	8.00	8.00	8.00	8.00	9.00	9.00	6.00	6.00	8.10
FE	9.00	10.00	7.00	9.00	4.00	5.00	3.00	4.00	2.00	3.00	5.60
AV-	9.00	6.00	4.00	4.00	5.00	5.00	1.00	0.00	1.50	2.00	3.75
AV+	10.00	10.00	9.00	9.00	6.00	6.00	5.00	8.50	7.50	3.00	7.40
HP-	10.00	8.00	10.00	8.00	8.00	6.00	7.00	6.00	7.00	5.00	7.50
HP+	9.00	4.00	1.00	1.00	2.00	1.00	2.00	5.00	3.00	1.00	2.90
R	9.00	9.00	9.00	7.00	8.00	5.00	7.00	8.00	7.00	4.00	7.30

Table D.3. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map basesWorkers16x16A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	8.00	6.00	6.00	5.00	9.00	4.00	4.00	1.00	3.00	3.00	4.90
FC	9.00	10.00	9.00	8.00	6.00	7.00	6.00	6.00	5.00	4.00	7.00
CE	10.00	9.00	9.00	7.50	8.00	7.00	8.50	6.00	5.00	6.00	7.60
FE	7.00	7.00	5.00	3.00	2.00	3.00	4.00	1.00	1.00	1.00	3.40
AV-	2.00	1.00	0.00	1.00	0.00	2.00	1.00	2.00	2.00	1.00	1.20
AV+	0.00	6.00	10.00	10.00	10.00	5.00	7.00	7.00	4.00	6.00	6.50
HP-	0.00	6.00	9.00	9.00	9.00	8.00	8.00	7.00	5.00	2.00	6.30
HP+	3.00	0.00	1.00	3.00	2.00	2.00	0.00	4.00	4.50	5.50	2.50
R	8.00	9.00	7.00	10.00	6.50	3.00	2.00	5.00	8.00	6.00	6.45

Table D.4. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map basesWorkers24x24A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	9.00	8.50	10.00	4.00	3.00	7.00	5.00	4.00	5.00	6.55
FC	10.00	10.00	9.00	10.00	9.00	8.00	10.00	8.00	6.50	8.00	8.85
CE	10.00	8.00	10.00	9.00	10.00	7.00	8.50	7.00	6.50	7.00	8.30
FE	9.00	10.00	9.00	7.00	7.00	6.00	5.00	3.00	1.00	0.00	5.70
AV-	1.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	2.00	0.40
AV+	0.00	7.00	10.00	10.00	10.00	8.50	9.00	10.00	10.00	7.00	8.15
HP-	1.00	7.00	10.00	10.00	10.00	10.00	9.50	10.00	9.00	8.00	8.45
HP+	0.00	0.00	0.00	2.00	2.00	1.00	2.00	3.00	2.50	2.00	1.45
R	10.00	9.00	10.00	10.00	8.50	7.50	7.00	5.00	6.50	7.50	8.10

Table D.5. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map basesWorkers24x24A Barrack											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	6.00	7.00	4.00	5.00	7.00	8.00	7.00	4.00	6.00	6.40
FC	9.00	10.00	9.00	10.00	9.00	10.00	9.00	8.00	8.00	9.00	9.10
CE	10.00	9.00	10.00	10.00	10.00	10.00	8.00	7.50	7.00	8.00	8.95
FE	7.00	9.00	9.00	6.00	8.00	4.00	2.00	5.00	1.00	1.50	5.25
AV-	3.00	5.00	0.00	0.00	0.00	0.50	0.00	1.00	1.00	4.00	1.45
AV+	2.00	6.00	10.00	10.00	10.00	9.00	9.00	8.00	8.00	8.00	8.00
HP-	5.00	3.00	9.00	8.00	10.00	10.00	9.00	9.00	9.00	6.00	7.80
HP+	4.00	0.00	0.00	0.00	0.00	0.00	2.00	3.00	5.00	4.00	1.80
R	8.00	9.00	8.00	9.00	8.00	10.00	7.00	8.00	8.00	7.00	8.20

Table D.6. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map basesWorkers32x32A											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	9.00	4.00	5.00	4.00	5.00	3.00	2.00	2.00	3.00	1.00	3.80
FC	8.00	7.00	9.00	9.00	8.00	7.00	6.00	8.00	2.50	6.00	7.05
CE	7.00	6.00	6.00	4.00	8.00	5.00	2.00	6.00	4.00	3.00	5.10
FE	7.00	9.00	5.00	8.00	7.00	2.00	4.00	3.00	1.00	3.00	4.90
AV-	2.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.40
AV+	3.00	5.00	5.00	10.00	6.00	9.00	8.00	7.00	5.50	7.50	6.60
HP-	7.00	2.00	10.00	10.00	9.00	7.00	7.00	8.00	8.00	4.50	7.25
HP+	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20
R	10.00	8.00	5.00	6.00	2.00	4.00	3.00	5.00	1.00	0.50	4.45

Table D.7. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map basesWorkersBarracks32x32											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	9.00	9.00	5.00	6.00	3.00	3.00	4.00	1.00	3.00	3.00	4.60
FC	8.00	9.00	8.00	7.00	6.00	6.00	6.00	4.00	4.00	4.00	6.20
CE	8.00	9.00	9.00	8.00	6.00	6.00	6.00	4.00	2.00	3.00	6.10
FE	5.00	5.00	3.00	5.00	5.00	2.00	1.00	0.00	0.00	0.00	2.60
AV-	2.00	1.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50
AV+	4.00	2.00	4.00	10.00	9.00	7.00	10.00	6.00	5.50	5.00	6.25
HP-	2.00	8.00	6.00	7.50	8.00	8.00	8.00	5.50	5.00	5.00	6.30
HP+	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R	9.00	8.00	7.00	7.00	6.00	1.00	4.00	4.00	1.00	3.00	5.00

Table D.8. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map BloodBath.scmB											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	10.00	10.00	10.00	10.00	9.00	8.50	4.00	5.00	3.00	7.95
FC	10.00	10.00	10.00	10.00	10.00	7.00	9.00	7.00	5.50	5.00	8.35
CE	10.00	10.00	10.00	10.00	10.00	10.00	7.50	7.00	7.00	5.00	8.65
FE	10.00	10.00	10.00	10.00	10.00	7.50	6.50	5.50	4.50	4.00	7.80
AV-	10.00	10.00	10.00	10.00	10.00	7.00	5.50	4.50	4.50	2.50	7.40
AV+	0.00	0.00	1.00	1.50	2.50	2.50	2.00	2.00	3.00	1.50	1.60
HP-	0.00	0.00	0.00	0.50	0.50	1.50	1.50	2.00	3.50	4.00	1.35
HP+	8.00	10.00	10.00	10.00	9.50	7.50	4.50	5.00	5.50	3.50	7.35
R	10.00	10.00	10.00	10.00	10.00	9.00	8.50	6.50	5.00	4.00	8.30

Table D.9. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

A3N vs. A1N - Map BloodBath.scmD											
Strategy	Unrestricted Set Size N										Avg.
	1	2	3	4	5	6	7	8	9	10	
CC	10.00	10.00	10.00	10.00	10.00	8.00	4.00	4.50	1.00	1.00	6.85
FC	10.00	10.00	10.00	10.00	10.00	8.00	8.00	5.00	5.00	5.00	8.10
CE	10.00	10.00	10.00	10.00	10.00	10.00	9.00	5.50	6.00	5.50	8.60
FE	10.00	9.00	10.00	9.00	10.00	7.00	6.00	4.50	3.00	3.00	7.15
AV-	9.00	6.00	9.00	7.50	5.50	4.50	4.50	3.50	1.50	2.50	5.35
AV+	0.00	0.00	1.00	1.00	0.00	1.00	0.50	0.50	0.50	1.50	0.60
HP-	0.00	0.00	0.00	0.00	0.50	0.50	0.50	3.00	1.00	0.50	0.60
HP+	7.00	6.50	7.50	5.50	3.50	4.50	3.50	3.50	1.50	2.50	4.55
R	10.00	10.00	10.00	10.00	10.00	7.50	7.50	5.00	3.00	2.50	7.55

Table D.10. Winning rate of variants of A3N against A1N in 10 matches. The rows depict different strategies (Str.) and the columns different unrestricted set sizes (N).

Appendix E

Detailed Final Experiment

This appendix shows each one of the matches compiled at section 6.5, detailed by map size.

Map basesWorkers8x8A																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	0.0	0.0	10.0	10.0	0.0	7.0	0.0	6.0	3.0	0.0	0.0	0.0	0.0	0.0	36.0
AHT	10.0	-	3.0	10.0	10.0	0.0	10.0	2.0	10.0	10.0	1.0	9.0	8.0	8.0	2.0	93.0
STT	10.0	7.0	-	10.0	10.0	2.0	10.0	6.0	10.0	10.0	2.0	9.0	5.0	6.0	2.0	99.0
RAR	0.0	0.0	0.0	-	0.0	0.0	0.0	0.0	6.0	3.0	0.0	0.0	0.0	0.0	0.0	9.0
HER	0.0	0.0	0.0	10.0	-	0.0	2.5	0.0	5.0	4.0	0.0	0.0	0.0	0.0	0.0	21.5
WOR	10.0	10.0	8.0	10.0	10.0	-	10.0	10.0	10.0	10.0	8.0	9.0	8.5	7.5	6.5	127.5
LIR	3.0	0.0	0.0	10.0	7.5	0.0	-	0.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	27.5
SCV+	10.0	8.0	4.0	10.0	10.0	0.0	10.0	-	10.0	10.0	3.0	5.0	7.0	4.0	0.0	91.0
SSS	4.0	0.0	0.0	4.0	5.0	0.0	6.0	0.0	-	6.0	0.0	0.0	0.0	1.0	0.0	26.0
PGS	7.0	0.0	0.0	7.0	6.0	0.0	7.0	0.0	4.0	-	0.0	0.0	0.0	0.0	0.0	31.0
NS	10.0	9.0	8.0	10.0	10.0	2.0	10.0	7.0	10.0	10.0	-	9.0	6.0	10.0	0.0	111.0
A1N	10.0	1.0	1.0	10.0	10.0	1.0	10.0	5.0	10.0	10.0	1.0	-	7.0	5.0	0.0	81.0
GAB	10.0	2.0	5.0	10.0	10.0	1.5	10.0	3.0	10.0	10.0	4.0	3.0	-	5.0	0.0	83.5
SAB	10.0	2.0	4.0	10.0	10.0	2.5	10.0	6.0	9.0	10.0	0.0	5.0	5.0	-	2.0	85.5
A3N	10.0	8.0	8.0	10.0	10.0	3.5	10.0	10.0	10.0	10.0	10.0	10.0	10.0	8.0	-	127.5

Table E.1. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map FourBasesWorkers8x8																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	7.5	0.0	7.0	8.0	0.0	7.0	0.0	4.0	1.0	0.0	0.0	0.0	0.0	1.0	35.5
AHT	2.5	-	0.0	9.0	10.0	0.0	9.5	0.0	1.0	2.0	0.0	0.0	0.0	0.0	0.0	34.0
STT	10.0	10.0	-	10.0	10.0	1.0	10.0	3.0	8.0	6.5	3.0	0.0	5.0	0.0	0.0	76.5
RAR	3.0	1.0	0.0	-	10.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	15.0
HER	2.0	0.0	0.0	0.0	-	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0
WOR	10.0	10.0	9.0	10.0	10.0	-	10.0	10.0	10.0	10.0	5.0	10.0	9.0	7.0	8.0	128.0
LIR	3.0	0.5	0.0	10.0	10.0	0.0	-	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	23.5
SCV+	10.0	10.0	7.0	10.0	10.0	0.0	10.0	-	10.0	10.0	4.0	10.0	7.0	5.0	5.0	108.0
SSS	6.0	9.0	2.0	10.0	10.0	0.0	10.0	0.0	-	7.0	0.0	1.0	0.0	0.0	0.0	55.0
PGS	9.0	8.0	3.5	9.0	10.0	0.0	10.0	0.0	3.0	-	0.0	0.0	0.0	0.0	1.0	53.5
NS	10.0	10.0	7.0	10.0	10.0	5.0	10.0	6.0	10.0	10.0	-	8.0	4.0	3.0	6.0	109.0
A1N	10.0	10.0	10.0	10.0	10.0	0.0	10.0	0.0	9.0	10.0	2.0	-	1.0	1.0	5.0	88.0
GAB	10.0	10.0	5.0	10.0	10.0	1.0	10.0	3.0	10.0	10.0	6.0	9.0	-	7.0	7.5	108.5
SAB	10.0	10.0	10.0	10.0	10.0	3.0	10.0	5.0	10.0	10.0	7.0	9.0	3.0	-	9.0	116.0
A3N	9.0	10.0	10.0	10.0	10.0	2.0	10.0	5.0	10.0	9.0	4.0	5.0	2.5	1.0	-	97.5

Table E.2. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map basesWorkers16x16A																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	0.0	4.0	10.0	5.0	0.0	4.0	0.0	5.0	5.0	10.0	4.0	1.0	2.0	4.0	54.0
AHT	10.0	-	6.0	10.0	10.0	2.0	10.0	9.5	10.0	6.0	0.0	3.0	3.5	7.0	0.0	87.0
STT	6.0	4.0	-	6.0	8.0	1.5	7.0	5.0	2.0	4.0	6.0	1.0	3.0	7.0	4.0	64.5
RAR	0.0	0.0	4.0	-	0.0	0.0	0.0	0.0	0.0	0.0	9.0	0.0	2.0	2.0	0.0	17.0
HER	5.0	0.0	2.0	10.0	-	0.0	0.0	0.0	3.0	1.0	6.0	0.0	0.0	0.0	0.0	27.0
WOR	10.0	8.0	8.5	10.0	10.0	-	10.0	10.0	8.0	10.0	4.0	3.0	10.0	9.0	4.0	114.5
LIR	6.0	0.0	3.0	10.0	10.0	0.0	-	0.0	7.0	8.0	10.0	5.0	0.0	0.0	7.0	66.0
SCV+	10.0	0.5	5.0	10.0	10.0	0.0	10.0	-	10.0	8.0	7.0	1.0	2.0	2.0	2.0	77.5
SSS	5.0	0.0	8.0	10.0	7.0	2.0	3.0	0.0	-	3.0	10.0	5.0	3.0	3.0	4.0	63.0
PGS	5.0	4.0	6.0	10.0	9.0	0.0	2.0	2.0	7.0	-	10.0	5.0	1.0	6.0	2.0	69.0
NS	0.0	10.0	4.0	1.0	4.0	6.0	0.0	3.0	0.0	0.0	-	2.0	4.0	7.0	0.0	41.0
A1N	6.0	7.0	9.0	10.0	10.0	7.0	5.0	9.0	5.0	5.0	8.0	-	6.0	8.0	2.5	97.5
GAB	9.0	6.5	7.0	8.0	10.0	0.0	10.0	8.0	7.0	9.0	6.0	4.0	-	5.0	2.0	91.5
SAB	8.0	3.0	3.0	8.0	10.0	1.0	10.0	8.0	7.0	4.0	3.0	2.0	5.0	-	0.0	72.0
A3N	6.0	10.0	6.0	10.0	10.0	6.0	3.0	8.0	6.0	8.0	10.0	7.5	8.0	10.0	-	108.5

Table E.3. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map TwoBasesBarracks16x16																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	4.0	1.0	10.0	5.0	5.0	0.0	9.0	4.0	5.5	2.0	2.0	2.0	2.0	1.5	53.0
AHT	6.0	-	0.0	2.0	9.0	5.0	0.0	9.0	1.0	3.0	0.0	1.0	6.0	2.0	0.0	44.0
STT	9.0	10.0	-	10.0	10.0	10.0	1.0	10.0	7.0	9.0	10.0	10.0	10.0	10.0	7.0	123.0
RAR	0.0	8.0	0.0	-	0.0	10.0	0.0	10.0	4.0	3.0	10.0	6.0	10.0	9.0	1.0	71.0
HER	5.0	1.0	0.0	10.0	-	10.0	0.0	8.0	6.0	8.0	10.0	7.0	7.0	9.0	1.0	82.0
WOR	5.0	5.0	0.0	0.0	0.0	-	0.0	10.0	1.0	0.0	0.0	0.0	2.0	0.0	0.0	23.0
LIR	10.0	10.0	9.0	10.0	10.0	10.0	-	10.0	9.0	10.0	10.0	10.0	10.0	10.0	10.0	138.0
SCV+	1.0	1.0	0.0	0.0	2.0	0.0	0.0	-	0.0	0.0	3.0	0.0	0.0	0.0	1.0	8.0
SSS	6.0	9.0	3.0	6.0	4.0	9.0	1.0	10.0	-	6.0	9.0	9.0	8.0	10.0	3.0	93.0
PGS	4.5	7.0	1.0	7.0	2.0	10.0	0.0	10.0	4.0	-	10.0	8.0	10.0	10.0	3.0	86.5
NS	8.0	10.0	0.0	0.0	0.0	10.0	0.0	7.0	1.0	0.0	-	3.0	9.0	7.0	0.0	55.0
A1N	8.0	9.0	0.0	4.0	3.0	10.0	0.0	10.0	1.0	2.0	7.0	-	9.0	8.0	0.0	71.0
GAB	8.0	4.0	0.0	0.0	3.0	8.0	0.0	10.0	2.0	0.0	1.0	1.0	-	2.0	0.0	39.0
SAB	8.0	8.0	0.0	1.0	1.0	10.0	0.0	10.0	0.0	0.0	3.0	2.0	8.0	-	1.0	52.0
A3N	8.5	10.0	3.0	9.0	9.0	10.0	0.0	9.0	7.0	7.0	10.0	10.0	10.0	9.0	-	111.5

Table E.4. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map basesWorkers24x24A																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	0.0	4.0	10.0	6.0	5.0	4.0	10.0	4.0	4.0	10.0	8.0	3.0	3.0	3.0	74.0
AHT	10.0	-	0.0	2.0	10.0	0.0	10.0	3.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	37.0
STT	6.0	10.0	-	4.0	8.0	4.0	0.0	3.0	2.0	3.0	9.0	3.0	0.0	1.0	1.0	54.0
RAR	0.0	8.0	6.0	-	0.0	5.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	29.0
HER	4.0	0.0	2.0	10.0	-	0.0	0.0	0.0	3.0	1.0	8.0	1.0	1.0	0.0	0.0	30.0
WOR	5.0	10.0	6.0	5.0	10.0	-	0.0	10.0	8.0	3.0	6.5	0.0	2.0	2.0	1.0	68.5
LIR	6.0	0.0	10.0	10.0	10.0	10.0	-	10.0	6.0	4.0	10.0	8.0	0.0	4.0	3.0	91.0
SCV+	0.0	7.0	7.0	10.0	10.0	0.0	0.0	-	1.0	0.0	9.0	1.0	0.0	0.0	3.0	48.0
SSS	6.0	9.0	8.0	10.0	7.0	2.0	4.0	9.0	-	7.0	10.0	9.0	0.0	4.0	1.0	86.0
PGS	6.0	10.0	7.0	10.0	9.0	7.0	6.0	10.0	3.0	-	10.0	8.0	0.0	2.0	3.0	91.0
NS	0.0	10.0	1.0	0.0	2.0	3.5	0.0	1.0	0.0	0.0	-	5.0	0.0	0.0	0.0	22.5
A1N	2.0	10.0	7.0	10.0	9.0	10.0	2.0	9.0	1.0	2.0	5.0	-	0.0	1.0	0.0	68.0
GAB	7.0	10.0	10.0	10.0	9.0	8.0	10.0	10.0	10.0	10.0	10.0	10.0	-	7.5	7.0	128.5
SAB	7.0	9.0	9.0	10.0	10.0	8.0	6.0	10.0	6.0	8.0	10.0	9.0	2.5	-	4.0	108.5
A3N	7.0	10.0	9.0	10.0	10.0	9.0	7.0	7.0	9.0	7.0	10.0	10.0	3.0	6.0	-	114.0

Table E.5. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map basesWorkers24x24A Barrack																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	9.0	0.0	10.0	10.0	10.0	2.0	10.0	6.0	6.0	10.0	7.0	4.0	3.0	2.0	89.0
AHT	1.0	-	0.0	7.0	9.0	3.0	0.0	5.0	2.0	0.0	0.0	0.0	0.0	1.0	0.0	28.0
STT	10.0	10.0	-	10.0	10.0	10.0	1.0	10.0	9.0	10.0	10.0	10.0	1.0	8.0	8.0	117.0
RAR	0.0	3.0	0.0	-	0.0	10.0	0.0	10.0	0.0	0.0	9.0	0.0	0.0	0.0	0.0	32.0
HER	0.0	1.0	0.0	10.0	-	5.0	0.0	10.0	3.0	0.0	1.0	2.0	0.0	0.0	0.0	32.0
WOR	0.0	7.0	0.0	0.0	5.0	-	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	12.0
LIR	8.0	10.0	9.0	10.0	10.0	10.0	-	10.0	10.0	6.0	10.0	7.0	2.0	3.0	7.0	112.0
SCV+	0.0	5.0	0.0	0.0	0.0	10.0	0.0	-	0.0	0.0	10.0	0.0	0.0	0.0	1.0	26.0
SSS	4.0	8.0	1.0	10.0	7.0	10.0	0.0	10.0	-	5.0	9.0	6.0	2.0	3.0	0.5	75.5
PGS	4.0	10.0	0.0	10.0	10.0	10.0	4.0	10.0	5.0	-	10.0	9.0	0.0	2.0	2.0	86.0
NS	0.0	10.0	0.0	1.0	9.0	10.0	0.0	0.0	1.0	0.0	-	3.0	0.0	0.0	0.0	34.0
A1N	3.0	10.0	0.0	10.0	8.0	10.0	3.0	10.0	4.0	1.0	7.0	-	0.0	1.0	0.0	67.0
GAB	6.0	10.0	9.0	10.0	10.0	10.0	8.0	10.0	8.0	10.0	10.0	10.0	-	5.0	4.0	120.0
SAB	7.0	9.0	2.0	10.0	10.0	10.0	7.0	10.0	7.0	8.0	10.0	9.0	5.0	-	1.0	105.0
A3N	8.0	10.0	2.0	10.0	10.0	10.0	3.0	9.0	9.5	8.0	10.0	10.0	6.0	9.0	-	114.5

Table E.6. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map basesWorkers32x32A																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	10.0	5.0	10.0	10.0	10.0	2.0	2.0	6.0	7.0	10.0	5.0	2.0	2.0	7.0	88.0
AHT	0.0	-	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
STT	5.0	10.0	-	9.0	9.0	8.0	0.0	3.0	8.0	6.0	10.0	10.0	0.0	5.0	7.0	90.0
RAR	0.0	10.0	1.0	-	0.0	10.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	31.0
HER	0.0	10.0	1.0	10.0	-	0.0	0.0	0.0	2.0	4.0	8.5	2.0	0.0	0.0	2.0	39.5
WOR	0.0	10.0	2.0	0.0	10.0	-	0.0	0.0	0.0	0.0	8.0	0.0	0.0	0.0	0.0	30.0
LIR	8.0	10.0	10.0	10.0	10.0	10.0	-	5.0	9.0	8.0	10.0	10.0	2.0	7.5	5.0	114.5
SCV+	8.0	10.0	7.0	10.0	10.0	10.0	5.0	-	9.0	7.0	7.5	7.0	4.0	8.0	2.0	104.5
SSS	4.0	10.0	2.0	10.0	8.0	10.0	1.0	1.0	-	5.0	10.0	8.0	1.0	4.0	4.0	78.0
PGS	3.0	10.0	4.0	10.0	6.0	10.0	2.0	3.0	5.0	-	10.0	8.0	1.0	4.0	4.0	80.0
NS	0.0	10.0	0.0	0.0	1.5	2.0	0.0	2.5	0.0	0.0	-	0.0	0.0	0.0	0.5	16.5
A1N	5.0	10.0	0.0	10.0	8.0	10.0	0.0	3.0	2.0	2.0	10.0	-	2.0	0.0	3.0	65.0
GAB	8.0	9.0	10.0	10.0	10.0	10.0	8.0	6.0	9.0	9.0	10.0	8.0	-	8.0	7.0	122.0
SAB	8.0	10.0	5.0	10.0	10.0	10.0	2.5	2.0	6.0	6.0	10.0	10.0	2.0	-	2.0	93.5
A3N	3.0	10.0	3.0	10.0	8.0	10.0	5.0	8.0	6.0	6.0	9.5	7.0	3.0	8.0	-	96.5

Table E.7. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map basesWorkersBarracks32x32																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	10.0	1.0	10.0	10.0	10.0	1.0	0.0	5.0	3.0	10.0	5.0	0.0	1.0	2.0	68.0
AHT	0.0	-	0.0	3.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	14.0
STT	9.0	10.0	-	10.0	10.0	10.0	3.0	10.0	10.0	9.0	10.0	10.0	1.0	9.0	10.0	121.0
RAR	0.0	7.0	0.0	-	0.0	10.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	27.0
HER	0.0	0.0	0.0	10.0	-	0.0	0.0	0.0	1.0	2.0	5.5	2.0	0.0	0.0	0.0	20.5
WOR	0.0	10.0	0.0	0.0	10.0	-	0.0	0.0	0.0	0.0	7.5	0.0	0.0	0.0	1.0	28.5
LIR	9.0	10.0	7.0	10.0	10.0	10.0	-	4.0	10.0	10.0	10.0	7.0	3.0	9.0	6.0	115.0
SCV+	10.0	10.0	0.0	10.0	10.0	10.0	6.0	-	10.0	10.0	10.0	10.0	7.0	9.0	3.0	115.0
SSS	5.0	10.0	0.0	10.0	9.0	10.0	0.0	0.0	-	5.0	10.0	9.0	0.0	4.0	2.0	74.0
PGS	7.0	10.0	1.0	10.0	8.0	10.0	0.0	0.0	5.0	-	10.0	8.0	1.0	3.0	1.0	74.0
NS	0.0	10.0	0.0	0.0	4.5	2.5	0.0	0.0	0.0	0.0	-	0.5	0.0	0.0	0.0	17.5
A1N	5.0	10.0	0.0	10.0	8.0	10.0	3.0	0.0	1.0	2.0	9.5	-	1.0	0.0	1.0	60.5
GAB	10.0	9.0	9.0	10.0	10.0	10.0	7.0	3.0	10.0	9.0	10.0	9.0	-	8.0	8.0	122.0
SAB	9.0	10.0	1.0	10.0	10.0	10.0	1.0	1.0	6.0	7.0	10.0	10.0	2.0	-	0.0	87.0
A3N	8.0	10.0	0.0	10.0	10.0	9.0	4.0	7.0	8.0	9.0	10.0	9.0	2.0	10.0	-	106.0

Table E.8. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map BloodBath.scmB																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	10.0	7.0	10.0	10.0	10.0	10.0	3.5	8.0	10.0	10.0	10.0	8.0	9.0	4.0	119.5
AHT	0.0	-	0.0	0.0	8.5	0.0	0.0	0.0	1.0	1.5	8.5	0.0	0.0	0.5	0.0	20.0
STT	3.0	10.0	-	10.0	10.0	10.0	10.0	8.0	10.0	9.0	10.0	10.0	1.0	5.0	6.5	112.5
RAR	0.0	10.0	0.0	-	0.0	10.0	0.0	0.0	2.0	3.0	10.0	10.0	0.0	0.0	0.0	45.0
HER	0.0	1.5	0.0	10.0	-	0.0	5.0	0.0	4.0	2.0	9.5	3.5	0.0	0.0	0.0	35.5
WOR	0.0	10.0	0.0	0.0	10.0	-	0.0	0.0	7.0	7.0	10.0	6.0	0.0	0.0	5.0	55.0
LIR	0.0	10.0	0.0	10.0	5.0	10.0	-	0.0	10.0	4.0	10.0	10.0	0.0	2.0	2.0	73.0
SCV+	6.5	10.0	2.0	10.0	10.0	10.0	10.0	-	6.0	6.5	10.0	10.0	5.5	7.0	2.5	106.0
SSS	2.0	9.0	0.0	8.0	6.0	3.0	0.0	4.0	-	4.0	10.0	9.5	0.0	0.0	0.0	55.5
PGS	0.0	8.5	1.0	7.0	8.0	3.0	6.0	3.5	6.0	-	10.0	10.0	0.0	2.0	0.0	65.0
NS	0.0	1.5	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	-	2.0	0.0	0.0	0.0	4.0
A1N	0.0	10.0	0.0	0.0	6.5	4.0	0.0	0.0	0.5	0.0	8.0	-	0.0	0.0	0.0	29.0
GAB	2.0	10.0	9.0	10.0	10.0	10.0	10.0	4.5	10.0	10.0	10.0	10.0	-	8.0	8.0	121.5
SAB	1.0	9.5	5.0	10.0	10.0	10.0	8.0	3.0	10.0	8.0	10.0	10.0	2.0	-	2.5	99.0
A3N	6.0	10.0	3.5	10.0	10.0	5.0	8.0	7.5	10.0	10.0	10.0	10.0	2.0	7.5	-	109.5

Table E.9. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.

Map BloodBath.scmD																
	PS	AHT	STT	RAR	HER	WOR	LIR	SCV+	SSS	PGS	NS	A1N	GAB	SAB	A3N	Total
PS	-	10.0	8.0	10.0	10.0	10.0	10.0	8.0	10.0	10.0	10.0	10.0	6.0	7.0	4.0	123.0
AHT	0.0	-	0.0	0.0	10.0	0.0	0.0	0.0	1.0	2.0	9.0	0.0	0.0	0.0	0.0	22.0
STT	2.0	10.0	-	5.0	10.0	5.0	5.0	1.5	7.0	6.0	10.0	9.0	3.0	2.0	7.0	82.5
RAR	0.0	10.0	5.0	-	5.0	5.0	0.0	0.0	4.0	1.0	10.0	8.0	0.0	0.0	0.0	48.0
HER	0.0	0.0	0.0	5.0	-	0.0	5.0	0.0	3.0	1.0	9.0	0.0	0.0	0.0	0.0	23.0
WOR	0.0	10.0	5.0	5.0	10.0	-	5.0	0.0	5.0	3.0	10.0	1.0	0.0	2.0	1.0	57.0
LIR	0.0	10.0	5.0	10.0	5.0	5.0	-	0.0	7.0	1.0	10.0	9.0	0.0	0.0	1.0	63.0
SCV+	2.0	10.0	8.5	10.0	10.0	10.0	10.0	-	7.0	6.0	10.0	10.0	2.0	9.0	4.0	108.5
SSS	0.0	9.0	3.0	6.0	7.0	5.0	3.0	3.0	-	6.0	10.0	7.0	3.0	3.0	1.5	66.5
PGS	0.0	8.0	4.0	9.0	9.0	7.0	9.0	4.0	4.0	-	10.0	10.0	0.0	3.0	2.5	79.5
NS	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	-	2.5	0.0	0.0	0.0	4.5
A1N	0.0	10.0	1.0	2.0	10.0	9.0	1.0	0.0	3.0	0.0	7.5	-	0.0	0.0	0.0	43.5
GAB	4.0	10.0	7.0	10.0	10.0	10.0	10.0	8.0	7.0	10.0	10.0	10.0	-	8.0	4.0	118.0
SAB	3.0	10.0	8.0	10.0	10.0	8.0	10.0	1.0	7.0	7.0	10.0	10.0	2.0	-	1.5	97.5
A3N	6.0	10.0	3.0	10.0	10.0	9.0	9.0	6.0	8.5	7.5	10.0	10.0	6.0	8.5	-	113.5

Table E.10. Winning rate of the row player against the column player in 10 matches played in 1 map. If a method wins all matches, then it has a winning rate of 10.