Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Using Backward Chained Behavior Trees to Control Cooperative Minecraft Agents

JUSTIN SALÉR

Stockholm, Sweden, 2023

Using Backward Chained Behavior Trees to Control Cooperative Minecraft Agents

JUSTIN SALÉR

Master's Programme, Machine Learning, 120 credits
Date: June 30, 2023

Supervisor: Petter Ögren
Examiner: Christian Smith

Swedish title: Användning av bakåtkedjade beteendeträd för att kontrollera samarbetande agenter i Minecraft

Abstract

This report presents a strategy to control multiple collaborative intelligent agents acting in a complex, versatile environment. The proposed method utilizes back-chained behavior trees and 1-to-1 task distribution. The agents claim a task, which prevents other agents in the system to start working on the same task. Backward chaining is an algorithm for generating reactive agents from a set of goals. The method was evaluated in Minecraft with Microsoft's Project Malmo API. Two different scenarios were considered. In the first one, a group of agents collaborated to build a structure. In the second one, a group of agents collaborated while gathering material. We propose and evaluate three algorithms with different levels of agent-cooperation and complexity (Algorithm 1, Algorithm 2, and Algorithm 3). The evaluation shows that backward chained Behaviour Trees (BTs) works well for multi-agent coordination in complex versatile environments and that adding 1-to-1 task distribution increases the efficiency of the agents when completing the experiment tasks.

Keywords

Behavior-Based Systems, Multi-Agent Systems, Behavior Trees, Minecraft

Sammanfattning

Rapporten presenterar en metod för styrning av en grupp kollaborativa intelligenta agenter agerande i en komplex dynamisk miljö. Den förslagna metoden använder sig av bakåtkedjade beteendeträd och 1-mot-1 uppgiftsdistribution, där en agent reserverar en uppgift vilket hindrar andra agenter att börja arbeta på samma uppgift. Bakåtkedjning är en metod som möjliggör generering av flexibla agenter utifrån en lista av mål och krav. Metoden utvärderades i två olika scenarion i tv-spelet Minecraft. Agenterna samarbetar i det första scenariot med att bygga en struktur och i det andra scenariot med att samla material. Vi föreslår och utvärderar tre algoritmer med olika nivåer av agentsamarbete och komplexitet (Algoritm 1, Algoritm 2, och Algorithm 3). Utvärderingerarna indikerar att bakåtkedjade beteendeträd fungerar bra för multiagentkoordination i komplexa dynamiska miljöer och att 1-mot-1 uppgiftsdistribution ökar agenternas förmåga att genomföra experimentuppgifterna ytterligare.

Nyckelord

Beteendebaserade system, multiagentssystem, beteendeträd, Minecraft

Acknowledgments

I want to thank Petter Ögren, my supervisor, and Christian Smith, my examinator. I also want to thank Mart Kartasev, who taught me about Malmo when we worked on a previous Minecraft project. Finally, I want to thank my wife, Wenzhao Liu, for her unwavering support.

Stockholm, June 2023
Justin Salér

Contents

List of Figures

List of Tables

List of Algorithms

List of acronyms and abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BT	Behaviour Tree
DWG	Default World Generator
EDBT	Event-driven Behaviour Tree
FSM	Finite State Machine
FWG	Flat World Generator
HRS	Hard Request Sender
HTN	Hierarchical Task Network
LOS	Line of Sight
MAS	Multi Agent System
ML	Machine Learning
NPC	Non Playable Character
PPA	Precondition-Postcondition-Action
PPA-BT	PPA Behaviour Tree
RH	Request Handler
RL	Reinforcement Learning
ROS	The Robot Operating System
SRS	Soft Request Sender

Chapter 1

Introduction

The project investigates how well (backward chained) Behaviour Trees (BTs) works for controlling collaborative intelligent agents. The experiments are evaluated in Minecraft, a video game popular due to its complex and versatile world. Intelligent agents make autonomous decisions to achieve their goals by observing their environment. In the thesis, each agent has their own BT that controls them. A BT is a planning paradigm commonly used for controlling Artificial Intelligence (AI) in video games or for robots. The intended outcome of the thesis was to alter backward chaining, a method used to generate BTs from a set of goals, to create collaborative BTs for agents that distribute tasks between them. The proposed solution generates a BT specialized in controlling an agent that collaborates with other agents using a near-identical BT in a shared environment. The agents collaborate by reserving distributable tasks through a global communication channel. If another agent has reserved a task, the agent will avoid that task and instead find an available task to work on. Not all tasks benefitted from being distributed. The agents achieved some objectives more efficiently when several agents worked on them together. Therefore, the solution also had to work for such tasks.

The project investigates how well backward chained BTs can control collaborative intelligent agents in Minecraft. An intelligent agent makes autonomous decisions to achieve a goal by observing its environment. A BT is a plan execution paradigm widely used for AI in video game development and robotics. Minecraft is a video game with a complex and versatile world. The intended outcome was a suggestion on extending backward chaining to generate collaborative BTs that uses a blackboard for distributing tasks between each other. Backward chaining is an algorithm to create BTs from a set of goals [1]. The blackboard is a hub for BTs to share variables. Like backward chaining, the extension generates a BT from a set of goals. The extension generates a BT specialized in controlling an agent that collaborates with other agents using a near-identical BT in a shared environment. The agents collaborate by reserving distributable tasks through the blackboard. The agents' BTs inspect whether a task is reserved and will work on other tasks if so. Not all tasks benefitted from 1-to-1 task distribution. Instead, the system achieved some objectives more efficiently when more than one agent worked on them concurrently. Therefore, the solution also had to be a good choice for those tasks.

This chapter offers an introduction to the report. The chapter starts with Section 1.1 presenting high-level background knowledge from topics that help give a deeper understanding of what this thesis is about; BTs, Multi Agent Systems (MASs), and Minecraft. It is followed by Section 1.2 presenting the problem. What issue did the project try to solve? Section 1.2 includes the research question that laid the foundation of the entire thesis. Subsequently, Section 1.3 goes deeper into the reason behind investigating the particular problem, explaining the value behind the research. Section 1.4 follows up with the project's concrete goals. Then, Section 1.5 shows how achieving the goals requires a tailor-made research methodology. Section 1.6 follows up with delimitations yielded primarily by project scope limitations and performance constraints. Finally, Section 1.7 presents the report's structure.

1.1 Background

The background behind the project was to investigate the abilities of backward chained BTs for controlling collaborative agents in the Minecraft environment. An improved understanding of systems with several agents acting in the same environment, MASs, aids many fields within AI and control systems. The project uses Minecraft as the experiment environment due to the game's versatility. Backward chained BTs are proficient in controlling autonomous agents in dynamic environments. Since many dynamic environments such as Minecraft utilize multiple agents, examining how well backward chained BT perform in a MASs is of great interest.

This section briefly introduces several relevant concepts. These are Minecraft, BTs and MASs. Later, chapter 2 presents these concepts in more detail.

1.1.1 Minecraft

There are several factors why Minecraft is an exceptional experiment environment for AI research. One key aspect is Minecraft's versatility in the shape of an exemplary combination of predictable and unpredictable elements. Another reason for doing AI research in Minecraft is a framework called Project Malmo, which makes it effortless for any AI system to control Minecraft agents.

Minecraft is a good evaluation tool for autonomous agents since it includes both complex, predictable features, such as what is needed to craft a particular item, and unpredictable features, such as what is required to adapt to the actions of an adversarial agent. In Minecraft, the ways to complete several of the game's goals are relatively fixed. Obtaining a specific item generally has the same preconditions. For example, crafting an item always uses the same components and always succeeds. Likewise, gathering an item always requires the same tool. Yet, there are other circumstances where the outcome of the task execution is uncertain. For example, consider a scenario where an agent tries to achieve a straightforward goal while an antagonist, another agent, or a hostile Non Playable Character (NPC) tries to stop it.

The BT control system used Project Malmo to communicate with Minecraft [2]. Project Malmo is a Application Programming Interface (API) for Minecraft developed by Microsoft. Project Malmo's primary use case is Reinforcement Learning (RL) research, but it also works well for other AI research.

1.1.2 Multi Agent Systems

Minecraft is a multiplayer game, which makes it a good setting for evaluating collaboration in a MASs. Multiple agents acting in the same environment can either cooperate or oppose each other. In a MAS, there can be individual goals for the agents and over-arching goals for the system. An overarching goal of a MAS with cooperative agents can be completing each agent's individual goals. Several fields utilizing AI and control systems benefit from evaluating how to design agents that collaborate to efficiently and accurately achieve the objectives of the MAS. Collaborating agents can accomplish goals difficult or impossible for a single agent. The benefits of collaboration can be seen in how humans, ants, and other animals work together on complex issues. For example, the great pyramids would be impossible for a single person to build.

However, creating cooperative agents takes work. Agents that cooperate are generally more complex than agents acting alone. Agents who switch between working alone and collaborating in groups are particularly complicated. A variable group size adds even more complexity. This thesis investigates how to create agents that work alone or in groups of any size.

In a MAS, there are many ways to divide control, where one method is centralized control. In centralized control, a single central system controls all agents. The system is either remote or local. A local central system has a leader agent controlling all agents. The central system observes the full MAS state, containing the states of all agents, and uses that to determine actions for each agent in the system.

Distributed and decentralized control are other common control paradigms, where each agent is autonomous and has individual observation and action states. The difference between distributed control and decentralized control is that distributedly controlled agents utilize a global communication system. These control paradigms are generally more scaleable than centralized control for controlling agents with individual roles or tasks.

While autonomous agents tend to act without considering other agents, the fact that several fields benefit from collaborating agents is becoming more apparent. For example, in robotics, there are several use cases for MASs, such as transporting goods in factory environments or sweeping large areas [3].

1.1.3 Behavior trees

Backward chained BTs efficiency in completing a set of goals while staying reactive to a versatile environment makes it an excellent choice for autonomous agents acting in a Minecraft environment. The predictable features make it easy to set the parameters of the backward chaining algorithm to achieve various Minecraft goals. Meanwhile, the reactivity of the BT paradigm supports handling the game's unpredictable features. This section presents BTs broadly, while Section 2.1 contains a more comprehensive overview.

While BTs are related to Finite State Machine (FSM), BTs are generally more reactive and modular [4]. Concerning reactivity, both BTs and FSMs act based on their current observation state. Nevertheless, while the current state FSMs depends on their previous state, the BTs continuously reevaluate their present state from the latest observations. Hence, in a BT, the current state is independent of the previous one, cultivating elevated reactivity. Regarding modularity, BTs is optimally modular [5]. FSMs, on the other hand, utilizes GOTO transmissions to increase reactivity. However, the GOTO transmissions negatively affect the modularity [4]. Designing FSMs involves a recurring trade-off between modularity and reactivity. However, for BTs, the two attributes synergize and bolster each other.

1.2 Problem

The problem was to create a scalable and generic method based on backward chaining to generate BTs for agents adapting to other agents' actions. Although manually designing behaviors and Precondition-Postcondition-Action (PPA) triplets often are sufficient to promote collaboration, it suffers from poor scalability. Therefore, the sought-after technique must be generic and transferable to promote scalability.

While manually creating collaborative PPA triplets is straightforward, they tend to overfit towards the experiment [4]. For example, assume there is a MAS with two agents, Agent A and Agent B, collaborating in constructing and transporting an object. Agent A's role is to prepare the item, and Agent B's is to transport it. Agent B should collect the item when Agent A informs that it has finished the preparation. It would have been easy to add a specific action, **Inform Agent B that the object is ready**, for Agent A to command Agent B, and a specific condition for Agent B, **Agent A has informed the object is ready**. This solution works well in this precise scenario, but overfitting to the particular problem causes continuous maintenance when doing any changes to the BT. Creating these solutions is fine for projects with few agents and goals, but projects with many sizeable BTs demand scaleable solutions.

The method should also be transferable to systems with already designed BTs. Ensuring that the method is transferable and extendable requires that the technique's performance does not change if the PPA triplets are modified. This restriction regards the logic inside the behaviors, the construction of the PPA triplets, and the order of the precondition, postcondition, and action sets. Oppositely, the same PPA triplets the extension uses should keep working with traditional backward chaining.

A summarized version of the research question is

> ”How can backward chained BTs provide a good solution for multi-agent coordination in complex adversarial environments?”

1.3 Purpose

Evaluating the capabilities of backward chained BTs for controlling cooperative agents provides knowledge useful for many fields utilizing control systems. BTs are common for AI in game development and robotics. An improved understanding of their capacity in the context of MASs is good for both fields [6] [7]. The proposed strategy is system-independent and easy to integrate with preexisting AI. Versatile planning algorithms are useful in any dynamic environment, particularly for controlling several collaborative agents. Basing the method on backward chained BTs grants a good understanding of their abilities, particularly their advantages, disadvantages, and limitations when controlling cooperative agents. The strategy was evaluated in the Minecraft game world to determine its performance in complex environments. It also demonstrates the backward chained BTs 's ability to control Minecraft agents. However, the results are good references for any state-based AI for collaborative agents. Reflections on the societal implications of the project can be found at the end of the report, in Section 8.4.

1.4 Goals

The method had three conditions:

- It should outperform classic backward chaining in cooperative MAS scenarios.
- The alterations should not significantly impact a single-agent scenario.
- The system should maintain or improve its ability to achieve its goals when adding another agent.

1.5 Research methodology

We evaluate collaborative extensions of backward chaining in Minecraft. The primary reference case is classic backward chaining. The expanded paradigm had to be better than classic backward chaining at controlling cooperative agents. It was essential to identify scenarios where classic backward chaining was expected to excel and use them as reference scenarios when designing the proposed paradigm. If the paradigm performs better or equally well as classic backward chaining in scenarios where it is difficult to match classic backward chaining, it is easier to state that the paradigm performs better than or as well as classic backward chaining in every scenario. Each agent should be able to complete different tasks simultaneously, so decentralized and distributed control was preferred over centralized control.

1.6 Delimitations

The project had a few delimitations, some induced by performance constraints, and others stemmed from project scope limitations.

- The agents were only evaluated in a few scenarios. The Minecraft environment is vast, with a humongous amount of possible scenarios. The agents completing enough scenarios indicate they can complete the entire game. The two scenarios covered the most fundamental gameplay.
- The agents were evaluated in the same environment during all trials in an experiment suite. It would have been interesting to randomize the environment for each trial. That was difficult due to environmental restrictions in the scenarios.
- Computational constraints limited the project to at most five agents. However, five agents still provide good insights into the performance and accuracy of the paradigm.
- The project was limited to Minecraft. Evaluating the method in other environments would have been good for posterity, but it was outside the project scope.

1.7 Structure of the thesis

This section presents the structure of the thesis.

Chapter 2 presents background knowledge to help better understand the thesis. The presented background knowledge includes overviews of BTs and MASs, as well as Minecraft and Project Malmo. Chapter 2 also showcases related works in research of BTs for controlling MASs, as well as other MAS experiments in Minecraft. Subsequently, Chapter 3 presents the scientific techniques utilized in the project. Chapter 4 is an interlude on handling tasks that several agents must simultaneously work on to complete. Then, Chapter 5 explains the proposed algorithm for expanding backward chaining to enable collaboration through 1-to-1 task distribution. After that, Chapter 6 presents the experiments created to evaluate the proposed method. Thereafter, Chapter 7 shows and discusses the experiment results. Finally, Chapter 8 presents the conclusions and possible future work.

Chapter 2

Background

The thesis makes use of a wide set of results within AI, specifically about BTs and MASs. This chapter presents three segments of background required for understanding the thesis; Section 2.1 BTs, Section 2.2 MASs, and Section 2.3 Minecraft and Project Malmo. The chapter also includes Section 2.4, which describes related works, specifically other research which has used BTs in MASs or used Minecraft as an experimental environment for analyzing MAS.

2.1 Behavior trees

This section familiarizes the reader with BTs. First, Section 2.1.1 presents a BT and explains their structure. After that, Section 2.1.2 presents backward chaining and how it generates BTs. Section 2.1.2 includes Algorithm 1, a technical outline of backward chaining.

2.1.1 Overview of behavior trees

A Behaviour Tree (BT) is a control system paradigm widespread in several areas that utilize AI. The BT is a modular tree of behavior nodes where each behavior node observes or interacts with the environment. A BT monitors the environment to determine how the agent should act. The nodes have different roles based on their position in the tree. The leaf nodes of a BT are called behaviors, and the branch nodes are called composites.

BTs is extensively used both for controlling robots and NPCs in video games and other virtual worlds. They gained prevalence from controlling the AI in successful video games, such as Halo and Bioshock [8] [9]. As a result, it has become a core feature in the top two game engines, Unreal Engine and Unity [10]. In addition, BTs are becoming more commonly used within robotics [11]. For example, they can control swarming kilobots [11].

A BT is a tree of behavior nodes for deciding how an agent should act based on its observation state. The structure of a BT is exceptionally modular. Each node in the BT observes and interacts with the environment. The controller ticks (executes) the BT at a constant frequency. In games and simulations, the frequency can be synchronized with the game render loop, but it is not a requirement. When the controller ticks the tree, the nodes are traversed in order, from top to bottom and left to right. A ticked node is in one of three possible states:

- **Success**.
- **Failure**.
- **Running**.

Two factors determine which nodes the controller ticks; the structure of the BT and the current state of the environment.

Behaviours

There are two commonly used types of behaviors. Conditions and actions.

Condition nodes observe the observation state to evaluate a condition. When satisfying the condition, the node's state is **Success**, and when the condition is dissatisfied, the node's state is **Failure**. An example of a condition is **Is hungry**.

Action nodes act in the environment. When ticking an action node, the agent performs an associated action. The outcome of the action decides the state of the action node. The state of the action node is **Success** when the action is successful, **Failure** when the action is unsuccessful and **Running** when the action is still in progress with an unknown outcome. The **Running** state is vital for the tree to perform actions that take a long time while preserving reactivity in case of sudden environmental changes. An example of an action is **Eat**.

Composites

The project utilized two types of composites. Sequences and fallbacks. The composite node type decides which children nodes to tick. Both sequences and fallbacks will tick at least one child node, starting with the leftmost one.

While both sequences and fallback keep ticking their children based on their outcomes, they react differently to the same child states.

Sequences keep ticking their children as long as their state is **Success**, but if the state of a child is **Running** or **Failure**, the sequence stops ticking its children and obtains that state.

Sequences are an excellent choice for actions with several preconditions. An example is a sequence with two children: a condition **Door is open** and an action **Enter room**.

Fallbacks keep ticking their children as long as their state is **Failure**. On the other hand, if the state of a child is **Running** or **Success**, the fallback stops ticking its children and obtains that state.

Fallbacks work well with postconditions. Ticking a fallback with a postcondition and an action ticks the action only when the postcondition is unfulfilled. An example of a fallback is a fallback with two children: a condition **Door is open** and an action **Open door**.

2.1.2 Backward chaining

Backward chaining is an algorithm for generating a BT to control an agent that can fulfill a set of goals, $\vec{G}$. the algorithm uses a set of PPA triplets, $\vec{\text{PPA}}$, to generate the BT. This section presents the backward chaining, first abstractly and then concretely.

Backward chaining goals

Backward chaining requires defining the goals $\vec{G}$ as either static or dynamic conditions. In this definition, a static goal is one that, once the goal obtains a **Success** state, the goal maintains that state. On the other hand, a dynamic goal toggles dynamically between the **Success** and **Failure** states. A dynamic goal can unexpectedly enter the **Failure** states even if it has been in the **Success** state for an extended time. Examples of a static goal are the agent having a particular item in its inventory or the agent having adjusted the game world, for example, by building a house or other structure.

An example of a good dynamic goal is $G_1 =$ **Is not attacked by an enemy**. For example, assume that there is a set of goals $\vec{G} = [G_1, G_2]$, where $G_2 =$ **Has built a house**. The accompanying action of G_1 is **Defeat enemy**. The goal G_2 has several accompanying actions, **Place block at position $\mathbf{p_i}$**, where $p_{i i} = \vec{p}$ are a list of positions. Placing blocks at all positions in $\vec{p}$ forms a house. An agent generated from $\vec{G}$ builds a house while staying ready for an enemy attack. When no enemies are present, the agent can steadily focus on building. However, when an enemy attacks, the agent instantly changes focus to defeating the offender. The entire BT is in a **Success** state only when the agent has finished building the house and no hostiles are attacking the agent.

The agent achieves the goals by their order in $\vec{G}$. For example, if the goals from the example above had been in the opposite order, $\vec{G}' = [G_2, G_1]$, then the agent would only attempt to get rid of the enemy after finishing building the house.

Backward chaining requires a set of PPA triplets, $\vec{\text{PPA}}$ to generate a BT. Each PPA triplet, PPA_i, consists of a set of actions, $\vec{A}_j$, a set of postconditions C_i^{post}, and a set of preconditions, $\vec{C}_i^{\text{pre}}$. Performing the actions fulfill the postconditions. Meeting the preconditions are required to perform the actions successfully.

The root of the generated BT, $\mathcal{T}_0$, is a sequence whose children are PPA Behaviour Trees (PPA-BTs) $[\mathcal{T}] = \mathcal{T}_{ii}$, where each tree $\mathcal{T}_i$ is designed to independently fulfill a goal G_i in $\vec{G}$.

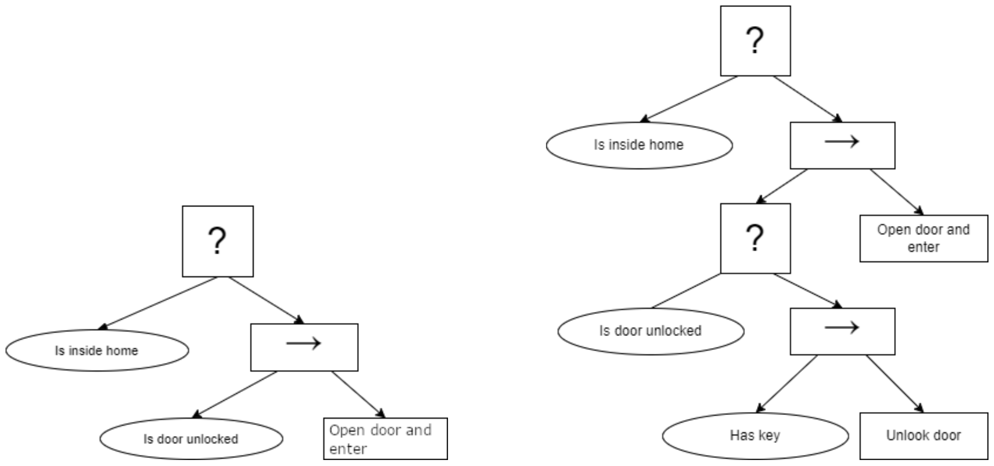
PPA Behavior trees

A PPA-BT is a subtype of BTs, designed to follow the rules of a PPA triplet. The mapping between goal G_i and PPA-BT $\mathcal{T}_i$ requires that G_i has similar criteria as a postcondition from a triplet in $\vec{\text{PPA}}$. When a controller ticks a PPA-BT, the system inspects the postconditions before executing the action. Executing the action is only valuable when the postcondition is unfulfilled. If that is the case, the controller also assesses the preconditions before executing the action nodes. If there are unfulfilled preconditions, the action nodes do not execute. Given that $G \in \vec{G}$ and $G \in \mathbf{C}_g^{\text{post}}$, we can construct the corresponding PPA-BT as

$$\mathcal{T}_g = \text{Fallback}(G, \text{Sequence}(\mathbf{C}_g^{\text{pre}}, \mathbf{A}_g)). \tag{2.1}$$

PPA-BTs are not necessarily convergent in an observation state. In this case, convergence means that a convergent BT, when ticked $n \to \infty$ times, eventually reaches a **Success** state. Whether a tree converges depends on the observation state. For example, Figure 2.1a displays a simple PPA-BT generated from PPA_1 in PPA triplets in Table 2.1.

The PPA-BT only converges after fulfilling all remaining preconditions $\mathbf{C}_g^{\text{pre}}$ in the entire environment space. Since convergence is environment-dependent, the most straightforward way to assess convergence is to evaluate whether the agent can complete the task in a test environment. The PPA-



(a) Before (b) After

Figure 2.1: PPA-BT before and after backward chaining

	C_i^{post}	C_i^{pre}	A_i
PPA$_1$	Is inside home	Is door unlocked	Open door and enter
PPA$_2$	Is door unlocked	Has key	Unlock door

Table 2.1: PPA triplets used for generating the example PPA-BTs in Figure 2.1a and Figure 2.1b.

BT depicted in 2.1a has narrow criteria for convergence, restricting its applicability to a limited set of states. It can control an agent that can open only unlocked doors and struggles in more complex circumstances. Nonetheless, the PPA-BTs in Figure 2.1a is an excellent foundation for using backward chaining to create a more advanced agent. Figure 2.1b portrays a BT generated from the tree in Figure 2.1a through backward chaining.

The backward chaining algorithm

Backward chaining is the process of extending a PPA-BTs to expand its set of manageable states by replacing each precondition C_{gi}^{pre} with a PPA-BT made from a PPA triplet with postcondition $C_{gj}^{\text{post}} = C_{gi}^{\text{pre}}$. Since the structure of the PPA-BT ensures that the controller always ticks postcondition nodes before ticking any precondition or action nodes, the substitution has no detrimental effect on the convergence of the tree. Each precondition $C_{gi}^{\text{pre}} \in \mathbf{C}_g^{\text{pre}}$ are iteratively replaced with a matching PPA-BT. Afterward, the backward chaining algorithm recursively exchanges the inserted PPA-BTs preconditions with new PPA-BTs.

The backward chaining algorithm expands the BT until the BT converges, or the backward chaining fails due to preconditions missing matching postconditions in PPA triplets. Since the backward chaining algorithm is often faster than evaluating the BT and additional backward chaining does not hurt the convergence, continuously assessing convergence during backward chaining is optional. To perform backward chaining until no mappable preconditions remain and then checking for convergence is generally faster. Algorithm 1 presents this variant of backward chaining in further detail.

In exchange for reduced generation time, Algorithm 1 produces larger BTs compared to backward chaining with continuous convergence assessment. If a PPA triplet has a precondition that is either a duplicate or a subcondition of another PPA triplet's precondition, depending on the structure of the backward chained BT, the precondition could end up being ticked only after the identical or encompassed precondition has been ticked with a **Success** status. As a result, the precondition is always in a **Success** state when ticked, and subjecting it to backward chaining does not increase the convergence of the BT. Whereas continuously assessing the convergence avoids subjecting such preconditions to backward chaining, Algorithm 1 subjects all mappable preconditions to backward chaining. Because of this, Algorithm 1 generates more extensive BTs that contain branches that are never executed [1].

Nevertheless, while the increased size affects the visual complexity and the backward chaining speed of the BT, it has a limited impact during the actual execution. The only difference in execution speed from replacing the precondition with a PPA tree comes from ticking a fallback node before the precondition. Additionally, all of the proposed solutions for extending backward chaining with 1-to-1 task distribution works equally well with Algorithm 1 and backward chaining with continuous convergence assessment. Algorithm 1 was chosen as the reference case for its simplicity and fast backward chaining.

[1] It is easy to maintain a list of already evaluated preconditions during backward chaining to prevent duplicate and dependent preconditions from being subjected to backward chaining. However, this method is insufficient for finding when a precondition is a subcondition of another precondition.

Algorithm 1 The backward chaining algorithm

```
Data: $\vec{PPA}$
Input: $\vec{G}$
Output: $\mathcal{T}_0$
1  $\mathcal{T}_0 \leftarrow$ Sequence
   foreach $G_i \in \vec{G}$ do
2      $S[i] \leftarrow$ Expand($G_i$)
3  Function Expand($C^{post}$):
4      $\mathcal{T}_{BC} \cup C^{post} \longleftarrow C^{post}$
       if $\exists P\vec{P}A(C^{post} \in \mathbf{C}^{post}_{PPA})$ then
5          PPA $\leftarrow P\vec{P}A(C^{post} \in \mathbf{C}^{post}_{PPA})$
           $(\mathbf{C}^{post}_{PPA}, \mathbf{C}^{pre}_{PPA}, \mathbf{A}_{PPA}) \leftarrow$ PPA
           $\mathcal{T} \leftarrow$ List<BT>()
           foreach $C^{pre}_i \in \mathbf{C}^{pre}_{PPA}$ do
6              $\mathcal{T}[i] \leftarrow$ Expand($C^{pre}$)
7          $\mathcal{T}_{BC} \leftarrow$ Fallback($C$, Sequence($\mathcal{T}, \mathbf{A}_{PPA}$))
           return $\mathcal{T}_{BC}$
8      else
9          return $C^{post}$
```

Figure 2.1b shows the tree for fulfilling the PPA triplet in Table 2.1 after backward chaining. The tree has grown from backward chaining and has a higher chance of convergence. If the agent starts with the key, the BT makes the agent complete its goal to enter the home. If the agent does not have the key, the tree does not converge. Then, crafting more PPA triplets enables backward chaining to expand the tree further.

The backward chaining algorithm iteratively expands each tree $\mathcal{T}_i$ in the root sequence $\mathcal{T}_0$. Backward chaining does not increase the convergence of any $\mathcal{T}_i$ and thus only has a possible impact on the convergence of $\mathcal{T}_0$.

2.2 Multi agent systems

Multi Agent System (MAS) are AI systems where several agents act in a single environment [12]. This section presents the MAS concept whi emphasizing autonomous cooperative agents. First, Section 2.2.1 presents strategies for distributing control in a MAS. Afterward, Section 2.2.2 presents methods to allocate roles between autonomous agents. These methods are also useable for distributing tasks, which is why they are interesting for the project.

2.2.1 Control distribution

The control in a MAS can be distributed in multiple ways [13]:

- Centralized control - A single central system controls all agents.
- Distributed control - Autonomous agents with a global communication system.
- Decentralized control - Fully autonomous agents with local decision making and without a reliable global communication system[1].

Pattern formation is a use case for MASs where decentralized control typically surpasses a centralized control system. Pattern formation implies that a group of agents moves together while maintaining a formation. Pattern formation is practical in both actual and fictional combat situations. Pattern formation is being investigated both within robotics and in video game environments [14] [15]. An example of pattern formation in robotics is a group of drones or land vehicles traveling together while avoiding inter-agent collisions. A large number of agents makes it difficult for a central system to control them all in real-time. A more straightforward approach is decentralized control, where each agent only needs to consider the positions of nearby agents [16]. Another benefit of decentralized control is that control systems that avoid non-interagent collisions, such as lidar, also can aid formation maintenance.

2.2.2 Role allocation

Role allocation is helpful for heterogenous agents, but defining their roles in advance is often challenging. Generally, formation maintaining MASs has predefined roles for each agent. However, for more complex MAS, predefining the roles considerably limits the efficiency. Then it is better to allocate the roles dynamically, which can be done in many different ways [17].

The easiest way to do dynamic role allocation is with a local or global broadcasting system, with many possible practices to choose from:

[1] A decentralized controlled MAS can have limited local communication.

- The agents use a minority game to allocate roles [18]. The agents continuously reevaluate their roles based on the roles of other agents in their proximity. This approach works exceptionally well in problems where the agents are heterogeneous, and a uniform role distribution is essential.

- The agents use SHAME to decide which agent performs which task. Here *SHAME* is a time-decaying variable representing the number of requests from other agents to perform the task from other agents [19]. The agents send *HELP* messages to each other, which increases the recipients' *SHAME* levels. The agent acts when *SHAME* exceeds a predefined threshold.

- The agents decide their role based on the full observation state [20]. The agents broadcast their observations, then each agent locally combines the observation states and evaluates how to distribute the tasks.

- The agents use a token-based system to distribute tasks between each other [21].

A lack of a broadcasting system enforces a belief system where agents choose their tasks based on their beliefs regarding other agents' states [17]. Belief system favors RL, and other Machine Learning (ML) methods [22].

The agent can use implicit roles instead of explicit ones when the roles are unclear or unknown. There are many ways to create implicit roles dynamically; for example, one is to utilize probabilistic methods during execution or pre-training [23] [24]. Another way to create implicit roles is through different ML algorithms, an example being agents reactively forming their role based on their capabilities with genetic programming [25]. Another example is to use neuroevolution to evolve role-based behaviors [26].

2.3 Minecraft

Minecraft is a popular multiplayer sandbox game [27]. Project Malmo is an AI framework for controlling a Minecraft agent [28]. The framework provides an API for observing and acting in Minecraft.

This section has two subsections. First, Section 2.3.1 presents a general overview of the Minecraft game logic, with a focus on the gathering-crafting loop that serves as a foundation of the entire gameplay. Second, Section 2.3.2 covers Project Malmo and how to use it for AI research in the Minecraft environment.

2.3.1 Gathering-Crafting loop

One of Minecraft's primary game loops is the Gather-craft loop [29]. The agent breaks the blocks making up the world to gather material. The agent then uses the collected material as components for crafting tools and other items. The synergy between gathering and crafting is a crafting-gathering loop, where the player crafts items to gather more material and gathers material to craft more items. With better tools, the agent can gather better material.

In Minecraft, there are five tiers of material[1], with higher-tier material enabling better tools, weapons, and equipment crafting. The five tiers sorted by rarity and strength are wood, stone, iron, gold[2] and diamond. Gathering diamond or gold requires an iron pickaxe, iron a stone pickaxe, and stone a wood pickaxe. Crafting a pickaxe requires three blocks of the primary material and two sticks [30]. Crafting sticks only requires wood, which does not require any specific gathering tool.

The agent needs weapons and equipment to fight enemies. Crafting weapons and equipment also require materials. Weapons are used for hunting and fighting NPCs and other players. Good weapons are essential since agents lose all their items if defeated. The agent can craft equipment to protect itself. Wearing equipment reduces the amount of damage taken. It can also build defensive structures, such as houses or walls. Building structures also require materials.

Minecraft has a combination of predictable and unpredictable objectives. For example, crafting items and mining materials are predictable objectives, while defeating enemies and searching for treasures are more unpredictable. The combination of determinism and stochasticity makes Minecraft an excellent environment for evaluating AI designs, such as BTs. For the predictable objectives, designing the PPA triplets is straightforward. For example, a simple mapping method can convert a crafting recipe for an item to a PPA triplet. In this example, a well-designed PPA triplet has an action to craft the item, a postcondition to have the item, and one precondition for having each component in the recipe.

In the same way, tool prerequisites foster a straightforward creation of PPA triplets for gathering materials. Table 2.2 contains some examples of PPA triplets. With a dynamic approach to create PPA triplets, backward chaining can generate a PPA-BT for obtaining any item. For example, Figure 2.2 shows the BT for an agent with the goal of crafting a stone pickaxe.

[1] Project Malmo requires Minecraft 1.11.2. Minecraft 1.11. which only have five tiers of material. A sixth tier, netherite, was added in Minecraft 1.16. [2] Gold is rare but fragile.

	C_i^{post}	C_i^{pre}	A_i
PPA_1	Has stone pickaxe	Has crafting table, Has 3 cobblestone, Has 2 sticks	Craft stone pickaxe
PPA_2	Has 3 cobblestone	Has wood pickaxe	Gather cobblestone
PPA_3	Has wood pickaxe	Has crafting table, Has 3 planks, Has 2 sticks	Craft wood pickaxe

Table 2.2: A subset of the PPA triplets used when generating a BT for obtaining a stone pickaxe.

The versatility of the Minecraft world works well with the reactivity of the BTs. In Minecraft, an agent can suddenly end up in a fatal situation involving water, lava, fire, or hostile NPCs. The agent must then react instantly and choose the correct action to save itself.

2.3.2 Project Malmo

The BT interacts with Minecraft through Project Malmo, an AI research framework [28] developed by Microsoft. The framework consists of both a library and a Minecraft plug-in. The library publishes an API for controlling an agent acting in Minecraft. The API enables observing and interacting in the game environment. Project Malmo also innately supports multiple agents.

Project Malmo includes a scenario environment generator that uses XML specifications for generating worlds with entities and agents. The specification defines the agents' initial conditions. The initial conditions include each agent's starting position, orientation, and inventory. In addition to the initial conditions, the specification also defines the agent's action and observation spaces.

Actions in Project Malmo

The action space can be defined in multiple ways in Project Malmo [28]. The primary decision is whether the agent should move through the world discretely or continuously.

Discrete movement is less faithful to the actual gameplay but is a reasonable simplification for agents that benefit from a smaller action space. Moving discretely, the agent teleports between the centers of the discrete block coordinates. The discrete movement works excellently for paradigms such as

RL, where the training time increases significantly with the size of the action space.

Continuous movement better simulates actual gameplay and works well with backward chained BTs. Moving continuously, the agent can perform motions such as strafing, pitching, yawing, and moving forward and backward. The movement commands include both a movement action and an intensity I. If $I = 0$, that particular movement stops. The continuous movement works well for backward chained BTs since they are highly reactive and do not require any training.

Besides moving, the agent has two main actions; hitting and interacting.

Hitting is used for attacking enemies and breaking blocks. An unequipped agent can deal damage to enemies and gather blocks. Nevertheless, tools increase the agent's gathering speed, and gathering capabilities and weapons increase damage output.

The agent can interact with both items in its inventory and utility blocks in its immediate environment. For example, a crafting table is a utility block enabling the agent to access all crafting recipes and craft any craftable item. Another utility block is a furnace for melting iron and gold ores into ingots. Ingots are required when crafting tools and weapons.

Project Malmo provides quick commands for crafting and melting to remove redundant complexity. Traditionally, when crafting, the agent carries the needed utility block in its inventory and places it in the environment before crafting an item. After the agent has finished crafting the item, it picks up the utility block and brings it around. With the quick commands, the agent can craft without placing the utility block. Instead, having the utility block and crafting components in the inventory is enough.

Hitting and interacting are also continuous in Project Malmo. For example, if an agent needs to break a wood block to gather wood, the controller must first order the agent to start attacking the block, and when the block breaks, the controller must command the agent to stop attacking.

Observations in Project Malmo

The observation state of a Project Malmo agent includes observations of the world, the internal state of the agent, and the agent's position and rotation relative to the world. The internal state of the agents covers attributes such as health and inventory.

A Project Malmo agent observes the world in three primary modes. The scenario specification specifies the available observation modes. The scenario specification also includes additional configuration of the observations. The three primary world observation elements are:

- Grid - Basic information of each block in a 3D Grid. The grid observation follows the agent's position or is stationary and placed in the game world. The size is configurable.
- Entities - Types and positions of nearby entities, for example, NPCs and pickupable items.
- Line of Sight (LOS) - The type and position of the block the agent is currently looking at.

The observation states have different uses. The relative grid observation is practical when looking for a specific material. While the LOS observation helps ensure that the agent does not start hitting before looking at the correct block or NPC. A static observation grid is practical when following the progress of building a structure. The entity observation state, on the other hand, aids in hunting hostile and non-hostile NPCs. The inventory observation informs the agent what items it has in its inventory and which items it has equipped. Ideally, the agent should hold a pickaxe when gathering stone or ores, an axe when gathering wood, a shovel when digging dirt, and a weapon when fighting enemies or hunting animals.

Multiple agents in Project Malmo

Minecraft with Project Malmo is a suitable environment for MAS experiments. Minecraft is a multiplayer game where players either cooperate or compete with each other. Some of the game's most compelling aspects are when players work together to search for rare items, defeat strong enemies, or construct large and complex structures. Project Malmo innately supports multiple agents. In the Project Malmo multi-agent implementation, one of them hosts the world. The other agents then connect to the world. The host is responsible for setting up the mission and restarting the server between each trial.

2.4 Related Works

This section presents and evaluates research that is related to this thesis. There is plenty of research about BTs and MASs, respectively. Additionally, combining the two paradigms have also been investigated priorly.

The section has three subsections; Section 2.4.1, Section 2.4.2, and Section 2.4.3. Section 2.4.1 sets the stage by presenting approaches for BT generation. Since the plan was to extend the backward chaining, getting an exhaustive understanding of various BTs generation techniques was beneficial. Subsequently, Section 2.4.2 presents ways to use BTs for controlling multiple agents. Finally, Section 2.4.3 shows related research about using Minecraft as a test environment for AI. Due to Minecraft's complex and versatile setting, several research articles have used the game for AI examination.

2.4.1 Behavior tree generation

BTs are traditionally manually constructed, but the requested scalability and modularity encourage several generative methods [31] [4]. The two most common BT generation methods are backward chaining and evolving BTs. The most significant difference between the two strategies is when generation happens. Backward chaining is a planning method and does not require any pre-training while evolving BTs continuously develops through training in a test environment.

This section first presents research about evolving BTs and showcases why the project instead uses backward chaining. After that, there is an overarching presentation about BTs as planning tools since a more detailed description of backward chaining is presented in the earlier Section 2.1.2.

Evolving behavior trees

Evolving BTs dynamically growing to fit a problem space has shown excellent results for controlling reactive agents in several fields, such as robotics and video games. An example from robotics is that evolving BTs successfully has learned how to control a swarm of resource-gathering kilobots [7]. Evolving BTs has also shown remarkable results in learning how to play different games, such as DEFCON, Super Mario, and Pac-Man [31] [32] [33].

To learn how to play Super Mario, the evolving BT used grammatical evolution [32]. Grammatical evolution is an evolution limited to a predefined syntax or grammar. The syntax restricts the shape of the evolving BT. The evolving BT that learned how to play Pac-Man had constraints similar to grammatical evolution, which also showed promising results [33].

Evolving BTs function well, although there are challenges in setting evolution rules such as reward definitions, fitness functions, and possible constraints.

Using planning tools to generate behavior trees

Whereas evolving BTs are versatile, stability is the typical trade-off of backward chaining and other planning-based generative methods.

For example, contrarily to evolving BTs, backward chaining has distinct generation and execution phases [1]. The separate steps make backward chained BTs more rigid and give them more apparent planning horizons. The stability made it straightforward to create agents that could continuously complete tasks in complex dynamic environments. The stability also enabled a good baseline when adding further MAS improvements. Section 2.1.2 describes backward chaining in more detail.

Another planning tool approach for BT generation is a two-step system proposed by Segura-Muros et al. [34]. First, a traditional planner, specifically a Hierarchical Task Network (HTN), creates an optimal strategy. Afterward, an algorithm converts the strategy to an executable BT. The authors used a robot simulation in The Robot Operating System (ROS) to evaluate the implementations, which showed strong results.

Evolving BTs and generating trees with planning tools are not necessarily isolated, as Styrud et al. show. They suggest a method that combines the two strategies [35]. They propose to first generate a BT with backward chaining. Afterward, the same BT evolves with genetic evolution. The genetic evolution algorithm learns from the problem state space and uses the learnings to reshape the BT. The method was successfully evaluated in a simulation with a robot learning how to organize and balance blocks.

2.4.2 Behavior trees for multi-agent systems

BTs works well for all manner of control of the MAS. A centralized control system has a single BT to control all agents. In decentralized or distributed control systems, the agents have individual BTs. The two procedures can be combined to promote cooperation in a distributed control system. Then, one or many managers BTs work together with the agent BTs. Other ways to support cooperation in decentralized or distributed control systems exist. The most prominent practice is to have the agents communicate with each other. However, if inter-agent communication is impossible or restricted, a belief system is enforced where the agents have to infer each other's actions.

Centralized control

A centralized control system with a single BT that manages all agents works well for a few identical agents but does not scale with many agents. With a large number of agents, the size of the state space grows uncontrollably [36]. Although BTs performs better than other AI control systems in large observation state spaces, enormous observation states still yield complex and unstable trees. The central BT needs to consider both the state space of each agent and the combined state space of the group. The tree quickly becomes complex and expanding it error-prone, especially when agents have distinct roles.

Distributed control

Distributed control can use agent-specific BTs together with the centralized manager BT. In this approach, the centralized BT focuses on high-level goals and the general observation state of the MAS environment. Meanwhile, the agents' BTs use detailed agent-specific information and high-level decisions from the manager BT to determine how the agent should act. This approach is typically more scalable than an alone manager BT.

Collendechaise et al. propose a variation of this approach, which utilizes two manager trees; a global task tree $\mathcal{T}_G$ and a task assignment tree $\mathcal{T}$ [1]. $\mathcal{T}_G$ sets high-level global tasks for the system, while $\mathcal{T}$ maps global tasks to local tasks. The agents then use local BTs $\mathcal{T}_{LI}$ for the actual environment interaction. The manager tree approach performs well for manually crafted trees, but due to the strong co-dependency between the manager tree and the agent trees makes it a bad fit for generated BTs.

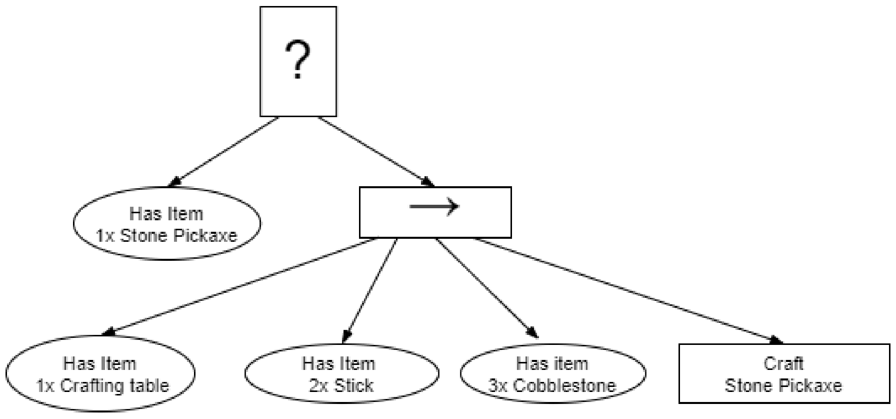
Another method is to skip the manager BT entirely and have the BTs communicate directly with each other. For example, in video games with a high amount of NPCs, each NPC does not need to act every game execution cycle. In these games, each NPC is controlled by an Event-driven Behaviour Trees (EDBTs). The difference between an EDBT and a traditional BT is that EDBTs are typically only ticked when receiving an event, where most events come from other EDBTs. Agis et al. propose constructing a EDBT with several behavior nodes adapted to handle communication. Request Handler (RH), Hard Request Sender (HRS) and Soft Request Sender (SRS) [6]. Jones et al. have a similar approach to Agis et al., but they apply it to evolving BTs [7]. Their report presents a BT with dynamically evolving sender actions and receiver conditions, communicating through a blackboard. Blackboard is a concept within BT methodology for sharing data between nodes and trees.

Decentralized control

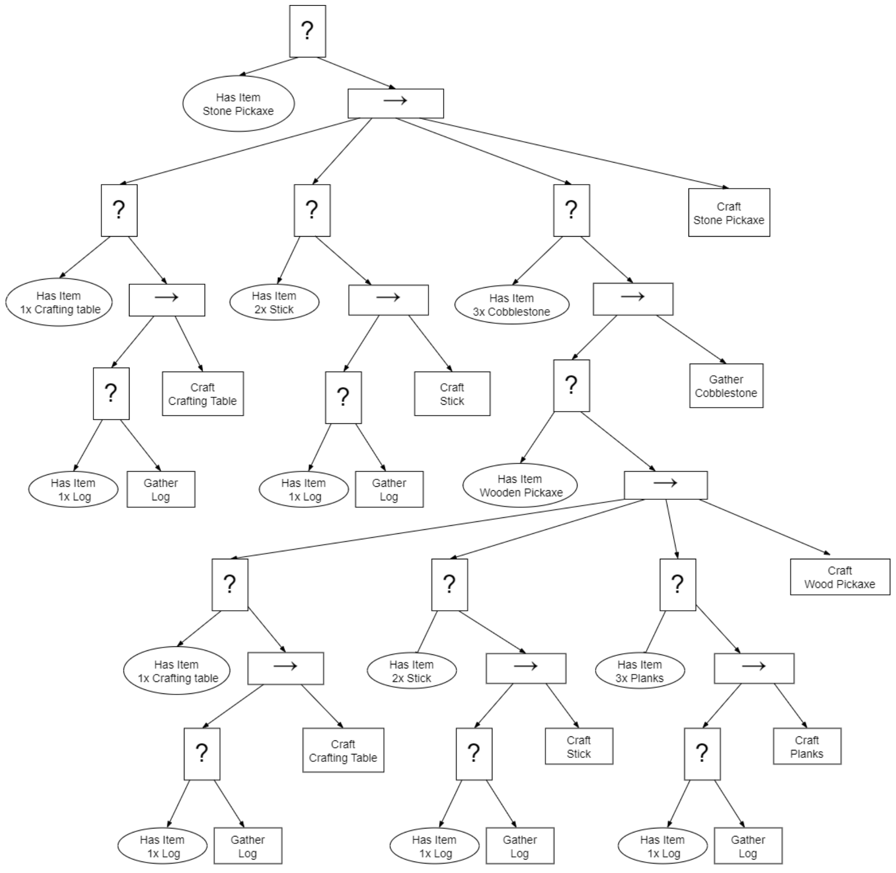
Decentralized control's absence of a stable communication method limits the agents to infer other agents' states and actions with a belief system. One way proposed by Ballagi et al. is using fuzzy signatures to deduce the intentions of the other agents [37] [38]. Fuzzy signatures are a paradigm where the agents utilize RL to learn the behavioral patterns of other agents. Agents can also use fuzzy signatures to guess the intentions of actual players.

2.4.3 Artificial intelligence in Minecraft

The team behind Project Malmo hosts annual RL competitions, where the participants create agents to complete a task in Minecraft. Most of the contests have focused on RL. The discrepancy between BTs and RL unfortunately makes the result from the two methodologies hard to compare. The theme of the first competition (2017) was collaboration [2]. The winner was an agent called HogRider, which used RL to learn how the other agents acted. HogRider used its learnings to create strategies for cooperation with agents of behaviors [39].



(a) Before backward chaining



(b) After backward chaining

Figure 2.2: Example of a backward chained BT for obtaining a stone pickaxe. The backward chained BT is generated by Algorithm 1. Some of the PPA triplets that were used are presented in Table 2.2
. Note that C_1 = **Has item 1x crafting table**, the first precondition of **Craft wood pickaxe**, is a duplicate of C_2 = **Has item 1x crafting table**, the first precondition of **Craft stone pickaxe**. Additionally, because of the structure of the BT, C_1 is only ticked when C_2 is in a **Success** state. Therefore, C_1 is always ticked as **Success**, and the backward chaining performed on C_1 is redundant. The observed

Chapter 3

Method

The primary gist of the research methodology was to devise an adequate algorithm with the aid of related research, set up a test environment in Minecraft, and evaluate the algorithm manually and automatically through an experimental test suite. The research was inductive, as the algorithm continuously evolved throughout the entire research period. The experiments were entirely digital, without any physical instruments or human participants. The proposed solutions, derived during the experiments, are presented in Chapter 4 and Chapter 5.

The chapter commences with Section 3.1 thoroughly presenting the research process. Section 3.2 then follows with an explanation of the research paradigm. Afterward, several sections about the experiments and data analysis follow. Section 3.3 describes how the data was collected for the thesis, Section 3.4 contains a breakdown of the experimental design, Section 3.5 describes how to make sure that the data is trustworthy, and Section 3.6 presents the data analysis techniques.

3.1 Research process

The full research process can be summarized as follows:

1. Literature studies to help identify which key mechanics of backward chained BTs are advantageous or detrimental for cooperation.

2. The creation of a test environment in Minecraft with one or many independent agents using backward chained BTs. This step included the manual design of PPA triplets. Testing the independent agent provides

insight into potential cooperation pitfalls. The independent agents were a baseline during the evaluation of the cooperative agents.

3. Definition and implementation of the proposed collaborative backward chaining algorithm in the test environment. Evaluation of the agents' ability to cooperate through 1-to-1 task distribution (where one agent is assigned to one task) [1]. The assessment considered two scenarios. In the first scenario, the agent built a structure in the game environment. In the second scenario, the agents had to gather material for crafting tools. The first scenario was designed to benefit from 1-to-1 task distribution. On the other hand, the second scenario was designed to show potential flaws and weaknesses of 1-to-1 task distribution, by limiting benefits from distributing tasks.

4. An evaluation of the method's advantages and disadvantages, done through comparison with the independent agents.

3.2 Research paradigm

The research paradigm was primarily inductive. At the start of the project, there were no clear answers to the research question, and the hypothesis still needed to be discovered during the literature study. This uncertainty made the first phase primarily inductive. The concept needed to be more detailed to interpret the process as deductive. The learnings during experimentation and evaluation constantly redefined the idea. The primary restrictions were that the solution had to fulfill the requirements proposed in Section 1.4.

3.3 Data collection

Data were collected during a set of experiments, with the number of trials having to be sufficient for the result to reach statistical significance.

The research was digital, without any human actors. The agents only interacted with other agents. The absence of questionnaires significantly reduced the time needed for evaluating the results. No questionnaires also mitigated any social and ethical concerns.

[1] The agents were a distributed control system, as they utilize a global broadcast system. However, a decentralized control system with a local communication system would require minimal changes in the methodology and have few, if any, downsides in the experimental setups

Additionally, contrarily to other AI paradigms, such as RL, the method did not require any pre-training. The backward chaining was very quick. The only significant time constraint stemmed from the requirement to experiment in real game time.

3.4 Experimental design

The method was evaluated in authentic Minecraft scenarios. Creation and manual evaluation were done the same way for both scenarios.

3.4.1 Scenario Creation

It was essential to create good scenarios that fulfilled certain conditions. Each scenario was to be evaluated in various group compositions. The two primary configuration parameters were the set of goals $\vec{G}$ and the list of PPA triplets $\vec{\text{PPA}}$.

The scenarios had a few requirements:

- The scenario had to be advanced enough to evaluate the proposed methods while preserving a reasonable level of complexity.
- The scenario required reactivity.
- The scenario should be possible for a single agent to complete.
- The scenario should be easier to complete when several agents collaborate.
- The scenario had to be feasible given the restrictions of Project Malmo.

Each scenario was evaluated in different compositions. All compositions should have comparable data and therefore needed to be able to complete the scenario. The compositions were:

1. A single agent running the independent policy.
2. A group of agents running the independent policy.
3. A group of agents running the collaborative policy.
4. A single agent running the collaborative policy.

3.4.2 Manual evaluation

Each scenario was manually evaluated in three steps:

1. Evaluation of the scenario with a single agent. The single agent evaluation required creating PPA triplets. The backward chaining algorithm used the PPA triplets to generate a BT for controlling the independent agent. The agent was evaluated by its ability to complete the scenario. When the single agent could finish the scenario, it proceeded to the next step.

2. Evaluation of the scenario with a group of disjoint agents. The group consisted of clones of the independent agent. There were two reasons to evaluate the disjoint group. First, to ensure that several agents could complete the scenario without obstructing each other. Second, the group was a reference point for possible collaboration improvements when designing the collaborative agents.

3. Evaluation of the scenario with a group of collaborative agents.

After the independent agents were done, the next step was to see whether a disjoint group of copies of the independent agents could complete the scenario. If the group couldn't, either the experiment required cooperation above the level of the independent agents or the design of the PPAs triplets was too poor and had to be revised. The possible flaws from lacking cooperation were used as inspiration when designing the collaborative agents.

After the single agent and the group of disjoint agents completed the objective, the next step was to evaluate the cooperative agents by comparing them with the independent agent and the disjoint group.

3.4.3 Experimental suite

After the implementation was stable, an experimental suite evaluated the methodology.

The experiments used two primary hyperparameters:

- n_{agents} - the number of agents.
- $\mathcal{S}$ - The strategy to generate the system.

The experiment also used a maximum time limit $t_{\text{max}} = 300s$ to avoid blocking the experimental suite if the agents became deadlocked and unable to complete the mission. The time limit was mainly impactful during the design of the PPA triplets. The PPA triplets were refined until both the agents running the independent policy, and the ones running the collaborative policy could continuously complete the experiments without reaching the time limit.

3.4.4 Software

The experiment used Python 3.7 with the libraries:

- **Project Malmo** - Framework for applying AI models in the Minecraft Game Environment
- **PyTrees** - Library for BTs. PyTrees does not include support for backward chaining.
- **Multiprocessing** - Built-in library for execution and communication for multiple processes.
- **Numpy** - Library for fast vector and matrix operations.
- **Pandas** - Library for data analytics.
- **Matplotlib** - Library for plotting.
- **Kivy** - Library for UI.

3.4.5 UI

Kivy was used to create a custom UI to mitigate issues during the scenario setup phase. The UI has two screens. The first screen is a configuration screen for setting hyperparameters of the test run, and the second screen is a dashboard for continuous feedback on the experiment.

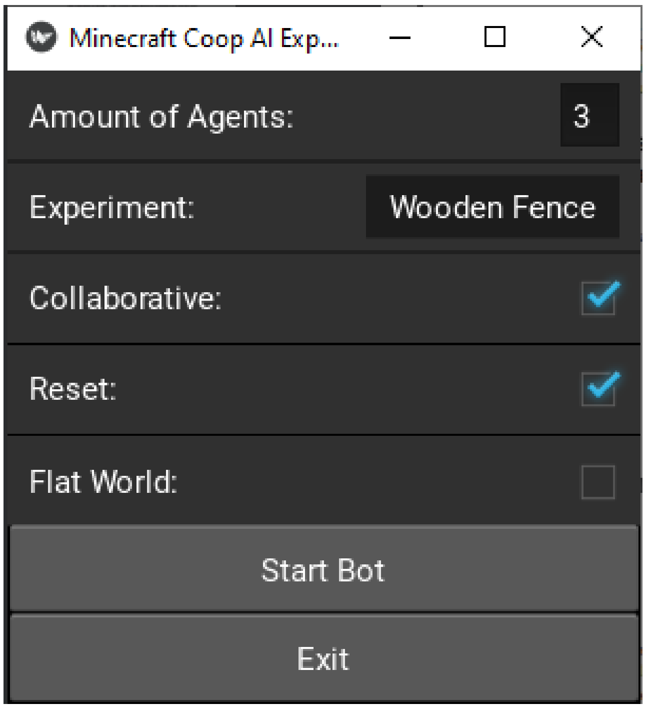
The dashboard has two fragments, both updated in real-time during execution. The first fragment shows the blackboard used by the agents to collaborate. The second fragment presents a map of agent positions. Figure 3.1 shows the UI screens.

3.5 Assessing reliability and validity of the data collected

Our approach was evaluated in several dimensions, both qualitative and quantitative. Some examples are efficiency, stability, and performance. While good performance is essential since the idea was to define an algorithm or methodology, ease of use and flexibility were just as important.

3.6 Data Analysis

The data analysis was limited to extracting the mean and standard deviation of the completion times. Pandas and Matplotlib are Python libraries that helped analyze the data and create graphs.



Amount of Agents: 3

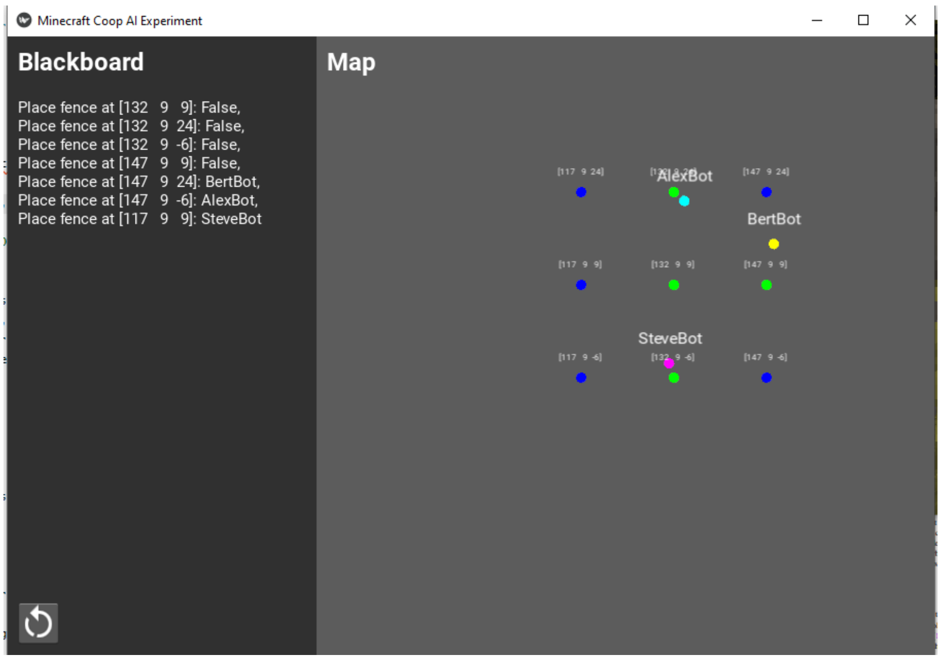
Experiment: Wooden Fence

Collaborative: ✓

Reset: ✓

Flat World: ☐

(a) Start Screen



```
Place fence at [132  9  9]: False,
Place fence at [132  9  24]: False,
Place fence at [132  9  -6]: False,
Place fence at [147  9  9]: False,
Place fence at [147  9  24]: BertBot,
Place fence at [147  9  -6]: AlexBot,
Place fence at [117  9  9]: SteveBot
```

(b) Dashboard

Figure 3.1: UI used to mitigate issues during scenario creation and manual evaluation.

Chapter 4

Reactive cooperation

Some goals, such as defeating an enemy, benefit from several agents working together to achieve them. In those situations, there are two critical factors to improve teamwork in urgent tasks requiring multiple agents. First, the urgent goals should be placed early in the goal set $\vec{G}$. For example, assume we have two goals

$$G_0 = \textbf{Hostile enemy is not nearby} \tag{4.1}$$
$$G_1 = \textbf{Have 100 wood}. \tag{4.2}$$

If $\vec{G} = [G_0, G_1]$, the agent will gather wood as long as no hostile enemies are close to the agent. However, if $\vec{G} = [G_1, G_0]$, the agent will completely ignore the hostile enemy and continue gathering materials while the enemy attacks it.

Second, agents cooperate better if they react swiftly when other agents need assistance [40]. In the context of BTs, this is done by creating conditions that consider other agents instead of only themselves [6]. For example, the goal G_0 from Equation (4.1) inspects whether a hostile enemy is close to the agent, which lets the agent defend itself. However, the goal

$$G_0' = \textbf{No hostile enemies near any an agent} \tag{4.3}$$

inspects whether a hostile enemy is close to any agent, enabling the agent to defend not only itself but also its allies. There are two primary methods to construct agents that continuously assess whether other agents need assistance and react quickly if so.

First, if endangered agents call for help when they need assistance, other agents can help them as soon as possible [19]. For example, the agents can use a PPA with the post-condition G_0 from Equation (4.1), with an action to broadcast when an enemy attacks them. The agents can use the blackboard to broadcast their need for assistance. Note that it is essential that the agents also broadcast when they are not in danger after the hostile enemy has disappeared. Then G_0', from Equation (4.3), can be a receiver node that checks the blackboard if any agents are in danger.

Second, agents that continuously observe each other are quick to help agents in need [41]. Be aware that the agents can only react when the agents are within observation range of each other. Additionally, when using G_0' from Equation (4.3), both the endangered agent and the threat need to be in the observation range of the agent. However, this requirement is trivial as long as the observation space is large enough to cover most situations where assisting is beneficial. If the adversary and the victim are too far from the possible support, interfering may be pointless. If the support tries to help when it is too far away, then when it gets close enough to protect the ally or defeat the enemy, the battle could have already ended.

Chapter 5

Collaborative backward chaining

We propose an algorithm (Algorithm 3) to generate a collaboratively backward chained BT that incorporates 1-to-1 task distribution to alleviate collaboration. In a MAS where the agents utilize traditionally backward chained BTs generated by Algorithm 1, the agents try to complete every task without considering the actions of each other. However, if the agents use the collaboratively backward chained from Algorithm 3, they prioritize collaboration through 1-to-1 task distribution. Algorithm 3 generates a BT that consists of a main collaborative BT and a backup independent BT. Whereas Algorithm 1, the traditional backward chaining algorithm, generates the backup tree, the algorithm to generate the collaborative main BT (Algorithm 2) is also a component of our proposal. Algorithm 2 generates a collaborative backward chained BT that communicates through a blackboard using sender and receiver nodes [1]. However, the BTs from Algorithm 2 does not have the backup tree from Algorithm 3, which causes a fundamental issue. The issue is that agents using BTs from Algorithm 2 idle when other agents have reserved all uncompleted tasks. In a way, adding the backup tree makes Algorithm 3 a refinement of Algorithm 2. The issue of the idling agents is significant enough that Algorithm 2 is less efficient than Algorithm 1 in some MAS scenarios, motivating the need for Algorithm 3. Some examples are when the scenario has few tasks, tasks with huge time differences, or tasks that several agents can work on simultaneously. Algorithm 3 mitigates the issue of idling agents by letting a traditionally backward chained BT serve as a backup by controlling agents when there are no unreserved tasks left.

[1] The blackboard contains a variable for each independent precondition denoting if any agent has reserved it.

This chapter starts with Section 5.1, a general overview of why task distribution is beneficial for MASs in the context of backward chained BTs. After that, Section 5.2 proposes a method to expand backward chaining to promote inter-agent cooperation. Section 5.3 announces some advantages and disadvantages of the collaborative backward chaining extension. In particular, a significant flaw appears when there are more agents than remaining unreserved tasks. However, as presented in Section 5.4, this deficiency can be removed by adding a refinement called ”backup”. Algorithm 3, ”collaboration with backup”, shown in Section 5.4 is the thesis' main proposal on how to expand backward chaining to create BTs for cooperative agents. Finally, Section 5.5 further proves the advantages of Algorithm 3 by highlighting some analytical execution time estimates of the completion times of agents using Algorithm 1, Algorithm 2, or Algorithm 3.

5.1 Task distribution

Task distribution benefits almost any MAS with many goals, including Minecraft. Agents with traditionally backward chained BTs (Algorithm 1) compete in achieving a set of tasks even when it would be more efficient to distribute the tasks between them. In Section 2.3.1, we show how the predictability of some goals in Minecraft makes them suitable for backward chained BT. The same predictability also makes MASs that act in Minecraft benefit from 1-to-1 task distribution.

Table 5.1 contains examples of PPA triplets from Minecraft, where some are distributable and others are not. Multiple agents can distribute the preconditions of PPA$_1$ between them, by simultaneously positioning wood at different target locations. The same can be said about PPA$_3$ if the agents can reallocate the items between them. However, PPA$_2$ is not distributable. There is no way to distribute the preconditions of PPA$_2$ between several agents. To perform the task a single agent must have an axe and be close to the tree. PPA$_4$ takes the indistributability one step further. It is not beneficial to distribute the pre-conditions. Contrarily, two agents need to work together to succeed.

The agents' awareness of each other's actions enables collaboration during the execution of the BTs. For example, assume that an agent, Agent A, wants to fulfill a condition C_1, which another agent, Agent B, is already trying to accomplish. Agent A then needs to evaluate which action is more beneficial from the perspective of the entire MAS. For example, should Agent A help Agent B fulfill C_1? Or should Agent A try to satisfy another condition, C_2, where C_2 is independent of C_1?

	C_i^{post}	C_i^{pre}	A_i
PPA_1	Have House	Wood at $\mathbf{X}_h$	None
PPA_2	Have iron pick axe	Have iron, Have sticks	Craft iron pick axe
PPA_3	Have wood	Close to tree, Have axe	Gather wood
PPA_4	Not attacked by hostile	Has hostile nearby, At least two agents	Defeat hostile

Table 5.1: Examples of PPA triplets. Here, $\mathbf{X}_h$ is a set of three-dimensional coordinates. Having wood blocks at all coordinates in $\mathbf{X}_h$ forms a house. PPA_1 and PPA_3 can be distributed. PPA_2 cannot. PPA_4 requires at least two agents.

5.1.1 Optimal task distribution

For efficient task distribution, it is often beneficial if the involved agents share a view on how the tasks should be distributed. There are several proposals on how to communicate to align the beliefs. We present two of the most straightforward methods. Both methods work well for smaller groups of agents, but they do not scale for larger groups. Because of this, the project did not focus on finding the optimal task distribution. Instead, we show that task distribution is beneficial, even with a suboptimal distribution.

Systems using optimal task distributions are prone to errors. When there is a discrepancy between the agents' ideas of the optimal task distribution, there is a high risk of errors for the entire system. For example, consider a system with two agents, Agent A and Agent B that has to fulfill a series of conditions. For Agent A to decide whether it should work towards fulfilling a condition C_1, it needs a belief system to determine whether Agent B or itself is the most suited to fulfill it given the current environment state. Simultaneously, Agent B needs to make the same assessment. Even if both agents utilize the same belief model, they have independent observation states. The agents using different information can cause a difference between the agents' optimal task distribution estimations. A potential problem is the agents ending up in a situation where their decisions depend on each other, and they end up in a loop where neither agent can complete the task.

Observation sharing

The first proposal is that the agents share their observation states [20]. Each agent then evaluates the observation state of all agents using an identical rule set. This method is not scalable for three reasons:

- The amount of inter-agent communications grows exponentially with the number of agents.
- As the size of the entire observation state grows linearly with the number of agents, combining the state spaces becomes exponentially more difficult with each additional agent.
- Every agent evaluating the same huge observation state of the MAS leads to massive overhead.

Avoiding these issues is a significant motivation for centralized control systems.

Belief sharing

The second proposal is that the agents share their beliefs [17]. However, the agents must share the beliefs for more conditions than condition C_1. To properly align the beliefs of the optimal task distribution, the agents must share their beliefs for all conditions the system can attempt to achieve. However, to share the beliefs, they need to be quantified. If the beliefs are quantified and shared, each agent can utilize the same rules on how to use the beliefs for task distribution. This method is also not scalable. The more agents that share their beliefs, the more tasks can be worked on in parallel and the more conditions the agents need to consider.

5.1.2 Suboptimal task distribution

Even distributing the tasks non-optimally is valuable and can be designed to avoid the scalability issues of optimal task distribution [42]. It also works well with backward chained BTs.

Distributing the tasks removes situations where the agents compete in completing the task before each other. Furthermore, each agent evaluating whether it should start working on and reserve a task is less complex than aligning beliefs. The proposed method is a decentralized system where the agents book tasks that they work on. When an agent is ready to fulfill a distributable condition C_i, it reserves it. Other agents see that the condition is already being worked on and continue with the next task.

The reservation system is a good fit for backward chained BT-controlled agents. In its foundation, backward chaining is a planning algorithm to fulfill a set of goals. If the goals are independent enough to distribute, the booking system transforms a group of independent agents into a collaborating task force. Also, if the preconditions in a PPA triplets are sufficiently disconnected, they can be distributed among agents.

5.2 Collaborative backward chaining

The primary part of our proposed method is to extend backward chaining to enable 1-to-1 task distribution by using receiver and sender nodes communicating through a blackboard. A blackboard contains variables shared between BTs. Each agent in the MAS utilizes near-identical BTs, which can communicate with each other through a shared blackboard B. The only difference between the BTs is the agent's name N_k, which the BT uses for reserving tasks.

Whereas it is possible to create collaborative agents with traditional backward chaining, as noted in Section 1.2, the proposed method had to explicitly work with non-collaborative PPA triplets. This is partly because collaborative PPA triplets tend to be less maintainable and the solution had to be straightforward to implement for systems utilizing backward chaining with a large set of PPA triplets.

This solution is a distributed control system. This is because the blackboard is a global communication system. However, the proposed method could also be modified to a decentralized control system by utilizing a local communication system instead. In a decentralized variant, the agents, instead of using a shared blackboard, would keep a local copy of their believed blackboard and update it as they receive updates from each other. If the range and stability of the local communication system are sufficient for the agents to communicate continuously, a decentralized control system performs as well as a distributed control system. However, limitations in the range or the stability can cause discrepancies in the agents' beliefs, and similar issues as with belief sharing (see Subsection 5.1.1) can occur.

The solution is also an example of the suboptimal task distribution mentioned in Subsection 5.1.2. Optimal task distribution could also have been done using a blackboard, which is as good as any global broadcasting system. However, as mentioned in Subsection 5.1.1, the blackboard would need to hold more information than just task reservations.

During our experiments, the agents stay relatively close to each other, so a local communication system might have been sufficient. However, as Minecraft only has a global communication system, it was not possible to test it. Nonetheless, this supports the notion that the approach could also be used by MASs without the possibility of global communication.

5.2.1 Blackboard communication behaviors

The agents communicate through a blackboard B. In BT methodology, a blackboard is a list of variables shared between behavior nodes or BTs. Since the blackboard is an inter-agent broadcasting system, the thesis refers to the variable names as ”channels”. We propose two new behavior nodes to alleviate communication through a blackboard: a sender node S and a receiver node R. Their implementations are inspired by Jones et al. [7].

The task distribution channel CH_i uses a unique identifier from the distributed condition C_i. All BT should be identical, so all agents' condition identifiers must be identical.

$\vec{V}$ contains unique identifiers N_k for each agent. In addition, it also has one more potential value. **None**, for when no agent are assigned to CH_i yet. When an agent has achieved the condition, the value is also **None**. The conditions' state depends on each agent's observation state. So as long as the agent estimates C_i to be incomplete and no one else is currently working on the task, the agent reserves the task. All agents' BTs are identical except for N_k.

Sender node

The sender node S is defined so that when ticked, it updates the value of the Channel CH with value V. Channel CH and V are parameters set to the sender node during the alternative backward chaining. Note that V does not have to be constant but can also be a variable that depends on the observation state. The sender node is constantly in **Success** state.

Receiver node

The receiver node R is defined to observe the blackboard for each tick. R and S use the same blackboard B and channels $\vec{\mathrm{CH}}$. The receiver node is an extension of a condition node. Each receiver node has two parameters; a channel CH and a list of accepted values $\vec{V}$. If $B(\mathrm{CH}) \in \vec{V}$, the receiver node will be in the state **Success**. Otherwise, its state will be **Failure**.

In addition to the receiver node, we propose $\bar{R}$, an inverse receiver node. In contrast to the receiver nodes, it has a list of unacceptable values $\vec{V}$. The inverse receiver node will be in the state **Success** if $B(\text{CH}) \notin \vec{V}$, and **Failure** otherwise.

5.2.2 Backward chaining extensions for collaboration

Here we present an extended backward chaining algorithm for creating the collaborative agents. For non-distributable tasks, the extension uses the standard backward chaining algorithm, but for distributable tasks, it uses a modification that facilitates 1-to-1 task distribution. The extended algorithm is presented in Algorithm 2.

The task distribution requires three specific blackboard nodes. Given a distributable task C_i, these are:

- An assigning sender node S_i. Ticking S_i informs the other agents that the agent reserves a condition C_i. The agents use their name N_k to reserve a task.
- An unassigning sender node S_i^0. Achieving C_i, ticks S_i^0 which sets $B(\text{CH}_i) = \textbf{None}$. Ticking S_i^0 makes C_i available for other agents. S_i^0 is required to maintain the reactivity of the BT. Even if the state of C_i is **Success**, environmental changes can make it **Failure** instead.
- An inverse receiver node $\bar{R}_i$. Ticking the node evaluates whether other agents are currently working on the task. If another agent has reserved the task, the node's state is **Success**. If not, it is **Failure**.

The inverse receiver node, $\bar{R}_i$, is used so the agent does not need to know the name of all other agents in the system. For example, an agent N_k has reserved C_i and therefore $B(\text{CH}_i) = N_k$. If an agent with the name N_l ticks $\bar{R}_i$, its state is **Success** if $N_k \neq N_l$, and otherwise its state is **Failure**. A receiver node however would require $\vec{V} = \vec{N} - N_l$, where $\vec{N} = N_{k_k}^{n_a}$ is the name of all agents in the system. An inverse receiver using $\vec{V} = [N_l, \textbf{None}]$ only need to know N_l and not entire $\mathcal{N}$.

Where applicable, a collaborative version replaces the PPA tree from traditional backward chaining. The design of the collaborative tree alleviates collaboration between agents. The collaboration happens by the agent broadcasting its tasks through blackboard nodes. Instead of $\mathcal{T}_g$ from equation (2.1), the collaborative version uses $\mathbf{T}'_i$, where

$$\mathbf{T}'_i = \text{Fallback}(\text{Sequence}(G_i, S_i^0), \bar{R}_i, \text{Sequence}(S_i, \mathbf{C}_g^{\text{pre}}, \mathbf{A}_g)). \tag{5.1}$$

Ticking this BT lets the agents review if a condition is reserved, and if it is unreserved, reserve it and attempt to achieve it. Ticking the tree, the agent first checks whether the goal G_i is fulfilled. If it is, the sender node S_i^0 unassigns the agent from the goal reservation. If the goal is unfulfilled, the receiver node $\bar{R}_i$ checks whether another agent already has reserved the goal. If the goal is free, the sender node S_i reserves the goal for the agent. Then if and when the pre-conditions $\mathbf{C}_g^{\text{pre}}$ are fulfilled, the agent performs the actions $\mathbf{A}_g$ in order to achieve G_i. If the goal is reserved, $\mathbf{T}'_i$ returns **Success**.

The extended algorithm uses $\mathbf{T}'_i$ for distributable preconditions and the traditional PPA-BT $\mathbf{T}_i$ for non-distributable preconditions. Both trees are required since all PPAs triplets do not have preconditions that can be fulfilled in parallel. For example, PPA$_3$ in Table 5.1 requires achieving the preconditions serially. In theory, the rules for serialization and parallelization can be so complex that the preconditions require a dependency graph. However, since a BT also can be interpreted as a dependency graph, these cases can be avoided by simply refactoring the PPA triplets. For simplification purposes, which tree to use is controlled by giving each PPA triplet a boolean variable for distributability, D. D is true when all preconditions are fulfilled in parallel and false otherwise. Equation (5.1) can be used with distributability D to extend Algorithm 1, demonstrating traditional backward chaining. Algorithm 2 is an extended version of Algorithm 1, which boosts cooperation through 1-to-1 task distribution using the sender and receiver nodes.

Figure 5.1 compares traditional and collaborative backward chaining for the goal set

$$\vec{G} = [G_1, G_2] = [\textbf{Is tree by the lake}, \textbf{Has treasure from temple}]. \tag{5.2}$$

Consider a MAS with two agents, Agent A, and Agent B, both controlled by either the traditional independent BT in Figure 5.1a or the collaborative BT in Figure 5.1b. If the agents use the traditional backward chained BT in Figure 5.1a, Agent A checks if there is a tree by the lake. If there is no tree by the lake, the agent tries to plant a tree there. Then, Agent B does the same thing. It checks whether there is a tree by the lake and, if not, also tries to plant the tree. First, after planting the tree, the agents start working towards the second goal, going to the temple to acquire the treasure. With the traditionally backward chained BTs in Figure 5.1a, the agents complete their goals in

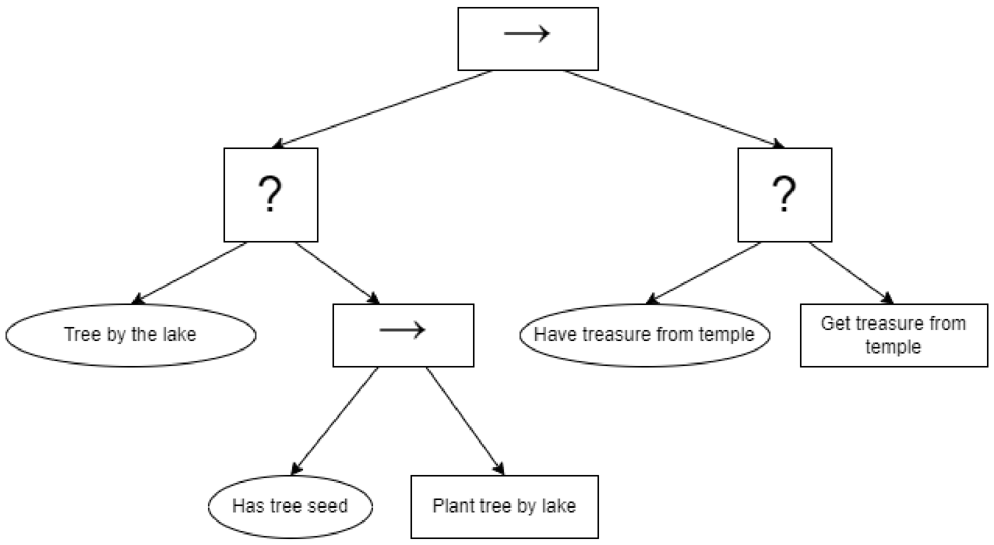
$$T_{12} = T_1 + T_2, \tag{5.3}$$

Algorithm 2 The collaboratively extended backward chaining algorithm

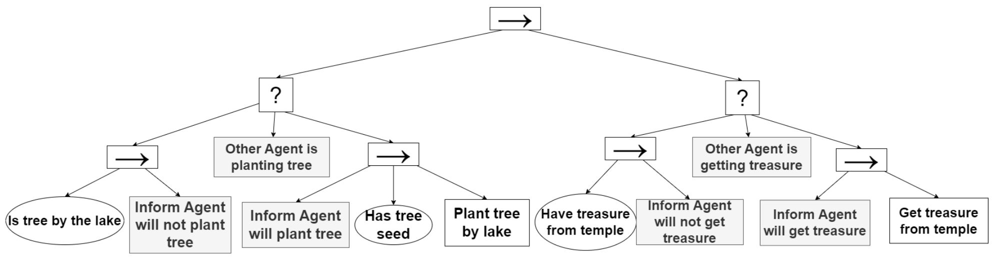
```
Data: $\vec{PPA}$
Input: $\vec{G}$
Output: $\mathcal{T}_0'$
10 $\mathcal{T}_0' \leftarrow$ Sequence
   foreach $G_i \in \vec{G}$ do
11     $S[i] \leftarrow$ ExpandCollaborative($G_i$)
12 Function ExpandCollaborative($C^{post}$):
13     $\mathcal{T}_{\text{Collab}} \cup \mathcal{T}_{\text{BC}} \cup C^{\text{post}} \longleftarrow C^{\text{post}}$
       if $\exists P\vec{P}A(C^{post} \in \mathbf{C}^{post}_{PPA})$ then
14         PPA $\leftarrow P\vec{P}A(C^{\text{post}} \in \mathbf{C}^{\text{post}}_{\text{PPA}})$
           $(\mathbf{C}^{\text{post}}_{\text{PPA}}, \mathbf{C}^{\text{pre}}_{\text{PPA}}, \mathbf{A}_{\text{PPA}}, D_{\text{PPA}}) \leftarrow$ PPA
           $\mathcal{T} \leftarrow$ List<BT>()
           foreach $C_i^{pre} \in \mathbf{C}^{pre}_{PPA}$ do
15             $\mathcal{T}[i] \leftarrow$ ExpandCollaborative($C^{\text{pre}}$)
16         if $D_{PPA}$ then
17             $\mathcal{T}_{\text{Collab}} \leftarrow \text{Sequence}(\text{Sequence}(G_i, S_i^0), \bar{R}_i, \text{Sequence}(S_i, \mathbf{C}_g^{\text{pre}}, \mathbf{A}_g))$
               return $\mathcal{T}_{\text{Collab}}$
18         else
19             $\mathcal{T}_{\text{BC}} \leftarrow \text{Sequence}(C, \text{Sequence}(\mathcal{T}, \mathbf{A}_{\text{PPA}}))$
               return $\mathcal{T}_{\text{BC}}$
20     else
21         return $C^{\text{post}}$
```

where T_1 is the time it takes to plant a tree, and T_2 is the time it takes to gather the treasure from the temple.

On the other hand, if the agents use the collaboratively backward chained BT in Figure 5.1b, Agent A starts working towards planting the tree. At the same time Agent A reserves G_1 on the blackboard to indicate that it is going to plant the tree. Then, when Agent B ticks their BT, it observes that G_1 is unfulfilled. However, it also sees from the blackboard that it is already reserved by Agent A. Because G_1 is already reserved, Agent B instead proceeds with checking if G_2 is unfulfilled and unreserved. Because it is, Agent B reserves



(a) Traditional backward chaining


→
?
?
→
Other Agent is planting tree
→
→
Other Agent is getting treasure
→
Is tree by the lake
Inform Agent will not plant tree
Inform Agent will plant tree
Has tree seed
Plant tree by lake
Have treasure from temple
Inform Agent will not get treasure
Inform Agent will get treasure
Get treasure from temple

(b) Collaborative backward chaining.

Figure 5.1: Traditional backward chaining (Algorithm 1) compared to collaborative backward chaining (Algorithm 2), for the goals $\vec{G}$ = [**Is tree by the lake**, **Has treasure from temple**]. The gray conditions are the receiver nodes, and the gray actions are the sender nodes.

G_2 and starts working towards accomplishing it by acquiring the treasure from the temple. With the collaboratively backward chained BTs in Figure 5.1b, The agents complete their goals in only

$$T_{12} = \max(T_1, T_2). \tag{5.4}$$

5.2.3 Reactive cooperation

We must also consider the opposite case, where agents must cooperate to fulfill a condition (see Chapter 4). An example from Minecraft is defeating monsters. The collaborative extension (Algorithm 2) can easily be adjusted to enable n-to-1 task distribution, with tasks reservable by multiple agents. However, without this adjustment, traditional backward chaining (Algorithm 1) is preferred, with the caveat that collaboration only occurs when both agents observe the threat. The reactive nature of BTs enables agents to work together in urgent situations. The ability of agents to work together in critical scenarios depends on the structure of the BT as well as the usage of the observation state. If an agent knows whether other agents need assistance, the agent can react swiftly to help. The agents can cooperate in critical situations by communicating or observing each other when possible [40].

Algorithm 2 can be altered for situations where multiple agents must collaborate to complete a single task. For example, assume that a task requires n_m agents to collaborate, where $n_m < n_a$. Algorithm 2 could be modified to allow several agents to reserve a condition together. This modification would only need minor adjustments to the blackboard nodes $\bar{R}_i$ S_i^0 and S_i.

If several agents cannot reserve a single task together, it is essential to refrain from distributing the tasks that benefit from several agents working together. In Algorithm 2, this means setting $D =$ False to the associated post-condition.

5.3 Advantages and disadvantages

The cooperative extension (Algorithm 2) has some areas where it performs better than traditional backward chaining (Algorithm 1) and others where it does not. Algorithm 2 is a more substantial improvement for tasks that do not benefit from being worked on by several agents simultaneously. Agents also benefit from 1-to-1 task distribution when the goals have substantial switching costs.

5.3.1 Shareable goals

A shareable goal is a goal that several agents can work on simultaneously for a faster completion time. If one or more goals are shareable, 1-on-1 task distribution, assigning one agent to each task, has a detrimental effect. The detrimental effect stems from agents idling when all remaining tasks are already reserved. On the other hand, if a goal is unshareable, there is a massive advantage in utilizing 1-to-1 task distribution. The benefit of 1-to-1 distribution is apparent when there are many more goals than agents, as long as the goals are unsharable. However, if the goals are sharable, there is a substantial drawback of 1-to-1 distribution, especially when there are more agents than goals.

Shareable goals can be defined by

Definition 5.1. Given that $T(G_s, n_a)$ is the completion time for completing a goal G_s with n_a agents. A goal is shareable if $T(G_s, 2) < T(G_s, 1)$.

Contrarily, unshareable goals can be defined by,

Definition 5.2. Given that $T(G_s, n_a)$ is the completion time for completing a goal G_s with n_a agents. A goal is unshareable if it is not shareable.

While this definition counts goals where $T(G_s, 2) \leq T(G_s, 1)$, one of the primary goals of the project, as stated in Section 1.4, is to find a method that maintains or improves its ability to achieve its goals when adding another agent. Then given this restriction and for ease of notation, we assume that for both our model and the reference model, $T(G_s, i{+}1) \leq T(G_s, i) \forall i \geq 1$. Thus in the context of this assumption,

Definition 5.3. Given that $T(G_s, n_a)$ is the completion time for completing a goal G_s with n_a agents. Also, adding another agent is never detrimental to completing the goals, that is $T(G_s, i + 1) \leq T(G_s, i) \forall i \geq 1$. Then, a goal is unshareable if $T(G_s, i + 1) = T(G_s, i) \forall i \geq 1$.

An example of a shareable goal is

$$G_s = \textbf{Agents have ten stone blocks}, \tag{5.5}$$

while an example of an unshareable one is

$$G_u = \textbf{Go and place a stone block at [10, 64, 10]}. \tag{5.6}$$

However, G_s is not sharable when agents independently decide to gather the same stone block. In the context of Project Malmo, G_s is shareable because the agents have independent observation states that let them collect the item closest to them. If the agents instead had shared observations and always attempted to mine the same block, gathering materials would not be a shareable goal. The shareability of the goals between agents when going for different blocks based on their observation state is also a form of task distribution. However, this distribution, which stems from the fact that the agents have personal observations, is harder to exploit than our proposal in Section 5.2.

The collaborative extension (Algorithm 2) does not always negatively impact the agents' efficiency and accuracy in scenarios with shareable goals.

It is limited to when the goal-switching cost (see below) is small or negligible. Consider the case where there is an external cost for switching between different tasks, such as when an agent walks between areas to gather different materials. Then even if the goals are shareable, it is beneficial to minimize the transiting time. Since agents using Algorithm 2 switches task less often than those using Algorithm 1 (Theorem 5.3), there is a possibility, when this cost is large, that Algorithm 2 outperforms 1 even for shareable goals. This occurs when the difference in time cost from additional tasks is greater than the cost of agents using Algorithm 2 idling when out of unassigned tasks.

5.3.2 Goal-switching cost

One of the reasons that non-optimal task distribution has a positive net effect is goal-switching cost. The goal-switching cost is the cost when an agent switches from one goal to another, originating from the extra effort of changing context. If the switching costs are significant, 1-to-1 task distribution is beneficial even for a set of shareable goals (goals that several agents can work on simultaneously).

The goal-switching cost is defined as follows:

Definition 5.4. The goal-switching cost is the additional time taken for an agent who completes a goal G_j after trying to complete a goal G_i, in contrast to an agent starting to complete G_j at the beginning. Note that the goal-switching cost is not constant and depends on the time it takes to go from G_i to G_j and how much time the agent has spent trying to complete G_i. Also, when there are several related goals, the cost can be negative. The mathematical notation we will use for goal-switching cost is $l_i^j(t_j^g)$, where $l_i^j(t_j^g)$ is the cost when switching from a goal G_i to another goal G_j after spending t_j^g seconds trying to achieve G_j.

The goal-switching cost originates in both environmental conditions and the structure of the BT. An example of a cost from environmental conditions is when an agent first has to run in one direction and then in another when gathering different materials. The impact of the BT structure on the goal-switching cost depends on the depth and differences of the preconditions. To switch to another task, the agent has to fulfill the preconditions. Hence, if the goals share any preconditions, the effort of switching focus decreases. Therefore, if the agent has to complete several preconditions to perform the task, leaving the task to another agent with better circumstances can be better.

In Section 5.5 we will prove that with significant goal-switching cost, for a set of unique shareable goals $\vec{G}_s$, a system of agents using collaborative backward chaining (Algorithm 2), $\mathcal{S}_C$, outperforms a system of agents using traditional backward chaining (Algorithm 1), $\mathcal{S}_I$.

5.4 Collaboration with backup

The final proposed solution is a refined collaborative extension that avoids idle agents by using a BTs which combines $\mathcal{T}_0$ from Algorithm 1 and $\mathcal{T}_0'$ from Algorithm 2, as

$$\mathcal{T}_0^* = \text{Sequence}(\mathcal{T}_0', \mathcal{T}_0), \tag{5.7}$$

$\mathcal{T}_0^*$ ticks $\mathcal{T}_0'$ until all goals are either satisfied or reserved by another agent. If all goals are reserved, the agents will not idle but instead, try to achieve the goals in order without considering which tasks are already booked by other agents.

Algorithm 3 portrays Equation (5.7) rewritten as an algorithm. By removing idling agents, collaboration with backup outperforms the collaborative backward chaining extension presented in Section 5.2. In situations where 1-to-1 task distribution works excellently, with unshareable goals and significant goal-switching costs, collaboration with backup (Algorithm 3) performs equally well as the collaborative extension (Algorithm 2). Since backup removes the risk of idling agents at the end of the experiment, it typically performs better than collaborative backward chaining without backup (Algorithm 2). These statements are proved analytically in Section 5.5 and are confirmed through experimental results in Section 7.1.

Furthermore, in scenarios with a low goal-switching cost and shareable goals where collaborative backward chaining (Algorithm 2) is less efficient than traditional backward chaining (Algorithm 1), collaboration with backup (Algorithm 3) performs equally well as traditional backward chaining (Algorithm 1).

5.5 Execution time estimations

This section contains analytical estimations of execution time comparing the three algorithms; traditional backward chaining (Algorithm 1), collaborative backward chaining (Algorithm 2), and collaborative backward chaining with backup (Algorithm 3). Subsection 5.5.1 contains proofs that Algorithm 3 has a shorter than or equal completion time as Algorithm 1 and Algorithm 2 for unshareable goals (Definition 5.3) and Subsection 5.5.2 contains proofs

Algorithm 3 The collaborative backward chaining algorithm with backup

```
   Data: $\vec{PPA}$
   Input: $\vec{G}$
   Output: $\mathcal{T}_0^*$
22 $n_G \leftarrow$ Length($\vec{G}$)
   $\mathcal{T}_0^* \leftarrow$ Sequence
   foreach $G_i \in \vec{G}$ do
23     $S[i] \leftarrow$ Expand($G_i$)
24 foreach $G_j \in \vec{G}$ do
25     $S[n_G + j] \leftarrow$ ExpandCollaborative($G_j$)
26 Function Expand($C^{post}$):
27     $\mathcal{T}_{\text{BC}} \cup C^{\text{post}} \longleftarrow C^{\text{post}}$
       if $\exists P\vec{P}A(C^{post} \in \mathbf{C}_{PPA}^{post})$ then
28         PPA $\leftarrow P\vec{P}A(C^{\text{post}} \in \mathbf{C}_{\text{PPA}}^{\text{post}})$
           $(\mathbf{C}_{\text{PPA}}^{\text{post}}, \mathbf{C}_{\text{PPA}}^{\text{pre}}, \mathbf{A}_{\text{PPA}}) \leftarrow$ PPA
           $\mathcal{T} \leftarrow$ List<BT>()
           foreach $C_i^{pre} \in \mathbf{C}_{PPA}^{pre}$ do
29             $\mathcal{T}[i] \leftarrow$ Expand ($C^{\text{pre}}$)
30         $\mathcal{T}_{\text{BC}} \leftarrow$ Fallback($C$, Sequence($\mathcal{T}, \mathbf{A}_{\text{PPA}}$))
           return $\mathcal{T}_{\text{BC}}$
31     else
32         return $C^{\text{post}}$
33 Function ExpandCollaborative($C^{post}$):
34     $\mathcal{T}_{\text{Collab}} \cup \mathcal{T}_{\text{BC}} \cup C^{\text{post}} \longleftarrow C^{\text{post}}$
       if $\exists P\vec{P}A(C^{post} \in \mathbf{C}_{PPA}^{post})$ then
35         PPA $\leftarrow P\vec{P}A(C^{\text{post}} \in \mathbf{C}_{\text{PPA}}^{\text{post}})$
           $(\mathbf{C}_{\text{PPA}}^{\text{post}}, \mathbf{C}_{\text{PPA}}^{\text{pre}}, \mathbf{A}_{\text{PPA}}, D_{\text{PPA}}) \leftarrow$ PPA
           $\mathcal{T} \leftarrow$ List<BT>()
           foreach $C_i^{pre} \in \mathbf{C}_{PPA}^{pre}$ do
36             $\mathcal{T}[i] \leftarrow$ ExpandCollaborative ($C^{\text{pre}}$)
37         if $D_{PPA}$ then
38             $\mathcal{T}_{\text{Collab}} \leftarrow$ Sequence(Sequence($G_i, S_i^0$), $\bar{R}_i$, Sequence($S_i, \mathbf{C}_g^{\text{pre}}, \mathbf{A}_g$))
               return $\mathcal{T}_{\text{Collab}}$
39         else
40             $\mathcal{T}_{\text{BC}} \leftarrow$ Sequence($C$, Sequence($\mathcal{T}, \mathbf{A}_{\text{PPA}}$))
               return $\mathcal{T}_{\text{BC}}$
41     else
42         return $C^{\text{post}}$
```

that the same holds for shareable goals (Definition 5.1). Finally, Subsection 5.5.3 shows that Algorithm 3 performs well in cases with a considerable goal-switching cost ((Definition 5.4).

5.5.1 Unshareable goals

Solutions using 1-to-1 task distribution (Algorithm 2 and Algorithm 3) favor scenarios with unshareable goals, where it is redundant for several agents to attempt completing the same goal (see Appendix B.1). We prove in this section that MASs using Algorithm 3 has a shorter than or equally short completion time as MASs using Algorithm 2. Appendix B.1 proves that systems with agents using Algorithm 2 have a shorter completion time with sets of unshareable goals than systems with agents using Algorithm 1. The proofs in this section and the ones in Appendix B.1 together state that Algorithm 3 provides better completion times than Algorithm 1 and Algorithm 3.

The proof that Algorithm 3 is an even better choice than Algorithm 2 for unshareable goals follows.

Theorem 5.1. *Given a set of unshareable goals, the completion time for agents using Algorithm 3 is shorter than for agents using Algorithm 2.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the same completion time $T(G_u, n_a) = t_u, \forall G_u \in \vec{G}_u, \forall n_a > 1$, then $T^u_{CB} \leq T^u_C$, where T^u_{CB} is the time it takes for $\mathcal{S}_{CB}$ to complete $\vec{G}_u$ and T^u_C is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_u$. Here $\mathcal{S}_{CB}$ is a system that uses collaborative backward chaining with backup (Algorithm 3), and $\mathcal{S}_C$ is a system that uses collaborative backward chaining without backup (Algorithm 2).

Proof. We separate this proof into two cases depending on the relationship between n_a and n_G. We know from Equation (5.7) that S_{CB} (using Algorithm 3) will work as a collaborative backward chaining tree until all tasks are reserved.

If $n_a \geq n_G$, the agents will separate into two packs.

- A collaborative group that uses 1-to-1 mapping to complete the tasks (Algorithm 2).
- A non-collaborative group that ignores the blackboard and goes through the tasks in order, as with traditional backward chaining (Algorithm 1).

At the start of the experiment, the first n_G agents will reserve a task each, and the remaining $n_a - n_G$ agents will see that all tasks are reserved and use the backup solution, which is to simultaneously go through the tasks in order, starting with the first goal. When any of the n_G agents complete their assigned task, they join this synchronously acting non-collaborative group. If any agent in the evergrowing non-collaborative group has better preconditions to complete a task than the assigned agent, that will reduce the completion time of the system so that $T^u_{CB} < T^u_C$. Worst case scenario, if none agents in the independent group have better preconditions than the assignee, $T^u_{CB} = T^u_C$. Thus, if $n_a \geq n_G$, then $T^u_{CB} \leq T^u_C$.

The advantages are similar for $n_a < n_G$, but it is noticeable first at the end of the goal set. While there are more than n_a available tasks, all agents will work collaboratively using 1-to-1 task distributions. However, when fewer than n_a free tasks are left, the absence of available tasks causes agents in $\mathcal{S}_C$ to idle. For $\mathcal{S}_{CB}$, on the other hand, the agents, which can not reserve a task, form a non-collaborative group. The final period, when less than n_a goals are left, shows the same pattern as the case where $n_a < n_G$. $\square$

5.5.2 Shareable goals

Solutions using 1-to-1 task distribution (Algorithm 2 and Algorithm 3) perform badly in scenarios with shareable goals, where it is redundant to distribute the tasks. It is possible to show that agents using traditional backward chaining (Algorithm 1) are faster at completing shareable goals, than agents using collaborative backward chaining (Algorithm 2) (see Appendix B.2). This section proves that collaborative backward chaining with backup (Algorithm 3) mitigates this disadvantage and performs as well as traditional backward chaining (Algorithm 1).

While our statements are for all shareable goals, we will limit ourselves to homogenously shareable goals for ease of calculation. For homogenously shareable goals, the completion time is inversely proportional to the number of agents n_a. As for unshareable goals, we will first look at a more naive case where the goals have the same completion time.

Definition 5.5. Given that $T(G_s, n_a)$ is the completion time for completing a goal G_s with n_a agents. A goal is homogenously shareable if $T(G_s, 2) < T(G_s, 1)$ and $T(G_s, i) \propto i^{-1}$.

The following theorem demonstrates that Algorithm 3 performs as well as Algorithm 1 for homogenously shareable goals.

Theorem 5.2. *Given a set of shareable goals, agents using Algorithm 3 have an equivalent completion time to agents using Algorithm 1.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with completion times $T(G_s, n_a) = T(G_s, 1)/n_a = T(G_s)/n_a, \forall G_s \in \vec{G}_s, \forall n_a > 1$, then $T^s_{CB} = T^s_I$, where T^s_{CB} is the time it takes for $\mathcal{S}_{CB}$ to complete $\vec{G}_s$ and T^s_I is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_s$. Here $\mathcal{S}_{CB}$ is a system using collaborative backward chaining with backup (Algorithm 3), and $\mathcal{S}_I$ is a system using traditional backward chaining (Algorithm 1).

Proof. Let us start by looking at the case where there are more agents than goals $n_a \geq n_G$, as that is where the idlers are the most prevalent. We can see in Equation (B.25) that for $\mathcal{S}_I$ the agents complete all goals in goal set $\vec{G}_s$

$$T_I(\vec{G}_s) = \sum_{G_s \in \vec{G}_s} \frac{T(G_s)}{n_a}. \tag{5.8}$$

As we have experienced, expressing the completion time for agents that utilize collaborative backward chaining is complex due to the scheduling problem-like nature.

Therefore, we will use an inductive approach. First, we will look at the case when $n_G = 1$. For $\mathcal{S}_I$, all agents will simultaneously try to complete the first task and we have

$$T_I(\vec{G}_s) = \frac{T(G_s)}{n_a}. \tag{5.9}$$

We have a different scenario but with the same outcome for $\mathcal{S}_{CB}$. The agents of $\mathcal{S}_{CB}$ sort themselves into two groups. The first group of n_G assigns themselves to a goal and tries to complete it. The second group roams, completing the goals in order (just as $\mathcal{S}_I$). In the case of a single goal, all agents will therefore try to complete the goal. Therefore, the completion time for $\mathcal{S}_{CB}$ is

$$T_{CB}(\vec{G}_s) = \frac{T(G_s)}{n_a}. \tag{5.10}$$

When we add a second goal, so we have $\vec{G}_s = [G_{s1}, G_{s2}]$,

$$T_I(\vec{G}_s) = \frac{T(G_{s1}) + T(G_{s2})}{n_a}. \tag{5.11}$$

T_I can still be expressed by Equation (5.9), but $T_{CB}(\vec{G}_s)$ is more complex to express. Let us assume that the goal set is sorted, with the fasted goal being the first one (this does not impact our results but makes calculation easier). Then at initialization, $\mathcal{S}_{CB}$ lets n_G agents assign themselves to a task, and $n_r = n_G - n_a$ agents are roamers that complete the tasks as a pack in order. We have $n_r + 1$ agents trying to complete G_{s1} and 1 agent working towards completing G_{s2}, giving us an expected starting pace,

$$\begin{aligned} T_{CB}(G_{s1}) &= \frac{T(G_{s1})}{n_r + 1} \\ T_{CB}(G_{s2}) &= T(G_{s2}). \end{aligned} \tag{5.12}$$

Because the goals are sorted by completion time, G_{s1} is completed first. If $T(G_{s1}) \leq T(G_{s2})$, it follows that $T(G_{s1})/(n_r + 1) < T(G_{s2})$. The first goal is completed at time $T(G_{s1})/(n_r + 1)$. Next, the agent assigned to G_{s1} joins the roaming pack, which tries to complete G_{s2}. Since $T(G_{s1})/(n_r + 1)$ has already passed, the remaining total completion time of G_{s2} is

$$T_{CB}(G_{s2}) = T(G_{s2}) - \frac{T(G_{s1})}{n_r + 1}. \tag{5.13}$$

We now have $n_r + 2$ agents working to complete G_{s2}, which means that the G_{s2}, and thus entire $\vec{G}_s$, is completed at the following time;

$$\begin{aligned} T_{CB}(\vec{G}_s) &= T_{CB}(G_{s2}) \\ &= \frac{T(G_{s1})}{n_r + 1} + \frac{T(G_{s2}) - \frac{T(G_{s1})}{n_r+1}}{n_r + 2} \\ &= \frac{T(G_{s1})}{n_r + 1} + \frac{(n_r + 1)T(G_{s2}) - T(G_{s1})}{(n_r + 1)(n_r + 2)} \\ &= \frac{(n_r + 2)T(G_{s1})}{(n_r + 1)(n_r + 2)} + \frac{(n_r + 1)T(G_{s2}) - T(G_{s1})}{(n_r + 1)(n_r + 2)} \\ &= \frac{(n_r + 1)T(G_{s2}) + (n_r + 2)T(G_{s1}) - T(G_{s1})}{(n_r + 1)(n_r + 2)} \\ &= \frac{T(G_{s2}) + T(G_{s1})}{(n_r + 2)} \\ &= \frac{T(G_{s2}) + T(G_{s1})}{n_a} \end{aligned} \tag{5.14}$$

The same simplification can be done for a larger number of n_G. Clearly, if $G_{s'}$ is the last goal in $\vec{G}_s$, then

$$
\begin{aligned}
T_{CB}(\vec{G}_s) &= \frac{\sum_{G_s}^{\vec{G}_s \setminus G_{s'}} T(G_s)}{n_a - 1} + \frac{T(G_s(n_G))) - \frac{\sum_{G_s}^{\vec{G}_s \ G_{s'}} T(G_s)}{n_a - 1}}{n_a} \\
&= \frac{\sum_{G_s}^{\vec{G}_s}}{n_a} \\
&= T_I(\vec{G}_s)
\end{aligned}
\tag{5.15}
$$

When there are more goals than agents, $n_a < n_G$, the roaming pack is formed first at the end of the experiment, but the same series calculation is applicable. □

5.5.3 Shareable goals with very high goal-switching costs

Our goal here is to assert that even for a set of shareable goals, if the goal-switching cost is high enough, a system, $\mathcal{S}_C$, using the collaborative extension (Algorithm 2 or Algorithm 3) can complete the goals faster than a system, $\mathcal{S}_I$, using traditional backward chaining (Algorithm 1).

Note that if there is a negative goal-switching cost (it is beneficial to perform two goals sequentially, this might happen if the agent learns from performing a task and then does the next one quicker than other agents would) Algorithm 3 can perform worse than Algorithm 1.

To not make our estimations too complex, we focus on the simplified case where the goal-switching cost is constant for all goals, $l_i^j = l$. [1].

Theorem 5.3. *Given a set of shareable goals, the completion time of agents that uses Algorithm 1 are more impacted by goal-switching cost than agents using Algorithm 2 and Algorithm 3.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with a constant goal-switching cost $l_i^j = l, \forall G_s \in \vec{G}_s$, then $L_C^s < L_I^s$, where L_C^s is the impact on the total time from goal-switching cost for $\mathcal{S}_C$ when completing $\vec{G}_s$ and L_I^s is the impact on the total time from goal-switching cost for $\mathcal{S}_I$ when complete $\vec{G}_s$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

[1] Our mathematical notation for goal-switching cost is l_i^j. Here l_i^j is the particular goal-switching cost when switching from a goal G_i to another goal G_j.

Proof. Each of the agents will work towards every goal. Also, since every agent is needed to complete the tasks, the goal-switching cost directly impacts the completion time. There is a total of $(n_G - 1)n_a$ switches, but each switch only adds a total time of l/n_a. Giving a total goal-switching cost as follows

$$L_I^s = (n_G - 1)l. \tag{5.16}$$

For $\mathcal{S}_C$, the amount of goal changes per agent is at best $\langle\frac{n_G}{n_a}\rangle$ changes and at worst $(n_G - 2)$ changes. However, as only the slowest agent's changes of their goal impact the system's completion time, the loss is only l. Hence,

$$L_C^s \in [\langle\frac{n_G}{n_a}\rangle l, (n_G - 2)l] \leq (n_G - 2)l < L_I^s. \tag{5.17}$$

$\square$

If we express the total times with goal-switching cost, the following statement can be made.

$$T_I^{s\prime} = T_I^s + L_I^s, \text{ and} \tag{5.18}$$

$$T_C^{s\prime} = T_C^s + L_C^s. \tag{5.19}$$

Then $T_C^{s\prime} < T_I^{s\prime}$ when

$$(L_I^s - L_C^s) > (T_C^s - T_I^s). \tag{5.20}$$

Finally, note that if there is a negative goal-switching cost, Algorithm 3 can perform worse than Algorithm 1. Negative goal-switching cost happens when any of the distributed goals benefits from being done by their order in the goal set [1]. An example is a set of goals $\vec{G} = [G_1, G_2]$, where $G_1 =$ **Any has five iron ingots** and $G_2 =$ **Any agent has an iron helmet** [2].

However, there are two reasons why negative goal-switching rarely impacts the performance of Algorithm 2 and Algorithm 3 relative to the performance of Algorithm 1.

- First, the most prevalent goals with negative goal switch distribution come from pre-condition and post-condition relationships. As these are embedded in the structure of BTs, the solutions are unaffected by these cases.

[1] 1-to-1 task distribution can avoid this issue or even benefit from it if the task distribution is optimized. More on this in Section 8.3. [2] Iron helmets are crafted by five iron ingots.

- Second, the distributability variable D is added to handle the limitation of distributing preconditions that are dependent on each other.

Chapter 6

Experiments

The proposed methods, Algorithm 3 and Algorithm 2 are evaluated in two Minecraft scenarios. The first scenario is ideal for the naive collaborative extension (Algorithm 2) and the second scenario highlights the algorithm's flaws, thus showcasing the advantages of the refined proposal adding backup (Algorithm 3). Section 6.1 presents a structure-building scenario, where the agents place fence posts in a grid-shaped pattern, and Section 6.2 presents a gathering scenario, where the agents gather materials for crafting stone pickaxes. Reactive cooperation is evaluated in a variant of the first scenario, which had a hostile entity attacking the agents.

The reason to evaluate a single collaborative agent is to ensure that the methodology has a loss-less effect on a single-agent system. This attribute is confirmed by verifying that collaborative backward chaining with backup method (Algorithm 3) performs equally well as traditional backward chaining (Algorithm 3) in the single-agent case.

The first primary scenario configuration parameter is the goals of the system, $\vec{G}$. The goals are explicit enough for the agents to use their limited observation space to assess whether they have achieved the goal.

The second and most crucial primary configuration parameter is the list of PPA triplets used by the backward chaining. The actions and conditions in the PPA triplets are manually defined. Therefore, their accuracy and efficiency directly impact the performance of the agent. The first step in creating each scenario is to design and evaluate them with a single independent agent. Each action and condition need to be stable to reduce bias.

Reactive cooperation is evaluated by adding a hostile entity to the first scenario. The first scenario was chosen since it requires the agents to visit the same points. If the enemy defeats one agent, the other agents need to take over the first agent's allocated task and likely pass the enemy while doing so. In the second scenario, the other agents had a higher chance of completing all goals without facing the enemy.

6.1 Fence post placement

The structure-building scenario evaluates the agents' ability to cooperate when placing fence posts in a grid pattern. The agents need to gather wood from nearby trees to craft the fence. The grid is a square 3×3 grid in the XZ plane. The agents are evaluated in two different procedurally generated test environments. The first test environment is an authentic world, generated by the built-in Minecraft Default World Generator (DWG), and the second test environment is a flat world, generated by the non-official Flat World Generator (FWG) [43].

6.1.1 Overview

The scenario evaluates the agents in two procedurally generated test environments generated by different procedural generation algorithms.

The first generation algorithm is the built-in Minecraft DWG, and the second generation algorithm is the non-official FWG [43].

The agents start in the savannah biome. In Minecraft, a biome is a local biotope that affects the parameters of the procedural generation. Different biomes have different distributions of blocks, altitude and density, and trees. The experiments take place in the savannah biome due to its tree density. In the savannah biome, the trees are sparse enough for the agents to move effortlessly without having to break obstructing trees but dense enough for the agents not to fight about resources.

Some worlds are generated by the Minecraft DWG to emulate a real game scenario, but the experiments also use the FWG due to limitations in the authentic worlds. Finding a good place in a world generated by the DWG to execute the scenario is challenging. The experiments require a large enough flat area for the agents to place the grid points. At the same time, the test environment needs enough trees for the agents to gather wood without straying too far from the fence post target locations. The grid points have the same altitude y to avoid excessive complexity. Therefore, the size of plateaus in the savannah biome put an upper threshold of Δ, the gap between neighboring posts. This upper constraint on Δ can be removed by evaluating the agents in a completely flat world. Project Malmo supports an unofficial FWG [43], which generates an entirely flat world. The FWG can also be configured to spawn structures and trees procedurally.

The DWG scenario and the FWG have some shared and distinct parameters. Both generation algorithms use a randomization seed s to reuse the same world in every trial. In the authentic world generated by the DWG, the experiments used the following hyperparameters:

- n_t - number of trials for each combination of n_a and $\mathcal{S}$.
- n_a - number of agents.
- $\mathcal{S}$ - the system used to control the agents; collaboration with backup (Algorithm 3), collaboration without backup (Algorithm 2), or traditional backward chaining (Algorithm 1).
- $\mathbf{P}_a = (\vec{p}_a^{\,i})_i^{n_{agents}}$ - The agents' initial positions.
- Δ - the vertical and horizontal distance between nearby fence posts.
- $\vec{p}_0 = (x_0, y_0, z_0)$ - the position of the fence post center.
- s - The randomization seed used by the DWG to generate the world.

However, in addition to the randomization seed, the FWG also uses a generator configuration string G. The generator configurator string restricts which blocks, biome, and structures the FWG generates. Thus, in the flat world generated by the FWG, the experiments used the following hyperparameters:

- n_t - number of trials for each combination of n_a and $\mathcal{S}$.
- n_a - number of agents.
- $\mathcal{S}$ - the system used to control the agents; collaboration with backup (Algorithm 3), collaboration without backup (Algorithm 2), or traditional backward chaining (Algorithm 1).
- $\mathbf{P}_a = (\vec{p}_a^{\,i})_i^{n_{agents}}$ - The agents' initial positions.
- Δ - the vertical and horizontal distance between nearby fence posts.
- $\vec{p}_0 = (x_0, y_0, z_0)$ - the position of the fence post center.
- s - The randomization seed used by the FWG to generate the world.
- G - The generator configuration string.

	C_i^{post}	C_i^{pre}	A_i
PPA_1	Is block B at position $\vec{p_i}$	Has item B equipped, Position $\vec{p_i}$ in reach	Place block B at $\vec{p_i}$
PPA_2	Has item I equipped	Has item I	Equip item I
PPA_3	Is position $\vec{p_i}$ in reach	None	Go to position $\vec{p_i}$
PPA_4	Has item I (craftable)	[Has item $m_i]{m_i}^{M(I)}$	Craft item I
PPA_5	has item I (gatherable)	Item I as pickup near	Pick up item I
PPA_6	Item I as pickup near	Block $B(I)$ in reach	Break block $B(I)$
PPA_7	Block B in reach	Block B is found	Go to block B
PPA_8	Block B is observable	None	Look for block

Table 6.1: PPA triplets used for the fence post placement scenario. These were used together with backward chaining to create the PPA-BT used for solving the scenarios. Here B is a block, I, an item, $\vec{p_i}$, a 3d position, $B(I)$, the block form of I, and $M(I)$, the material needed for crafting I.

6.1.2 Backward chaining

The initial goal set in the fence post placement,

$$\vec{G} = \{G_p(\vec{p_i})\}_i^{3\times 3}, \text{ where} \quad (6.1)$$

$$G_p(\vec{p}) = \textbf{Is block fence post at position } \vec{\mathbf{p}} \text{ and} \quad (6.2)$$

$$\{\vec{p_i}\}_i^{3\times 3} = \left\{ \{(x_0 + \Delta k_x, y_0, z_0 + \Delta k_z).\}_{k_x \in \{-1,0,-1\}} \right\}_{k_z \in \{-1,0,-1\}}. \quad (6.3)$$

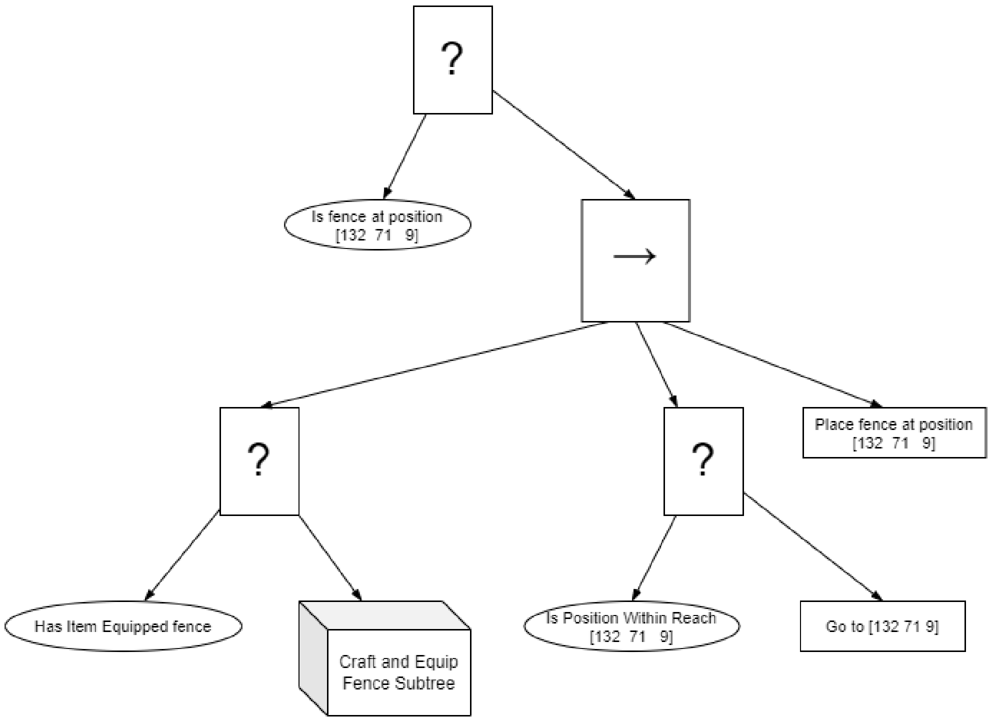
The goals $\vec{G}$ are unshareable, facilitating optimal conditions for collaborative backward chaining. In this scenario, only the goals in $\vec{G}$ are distributed through 1-to-1 task distribution.

Table 6.1 shows the PPA triplet used by the fence post placement scenario. Figure A.1 shows the full backward chained tree for a single fence post placement.

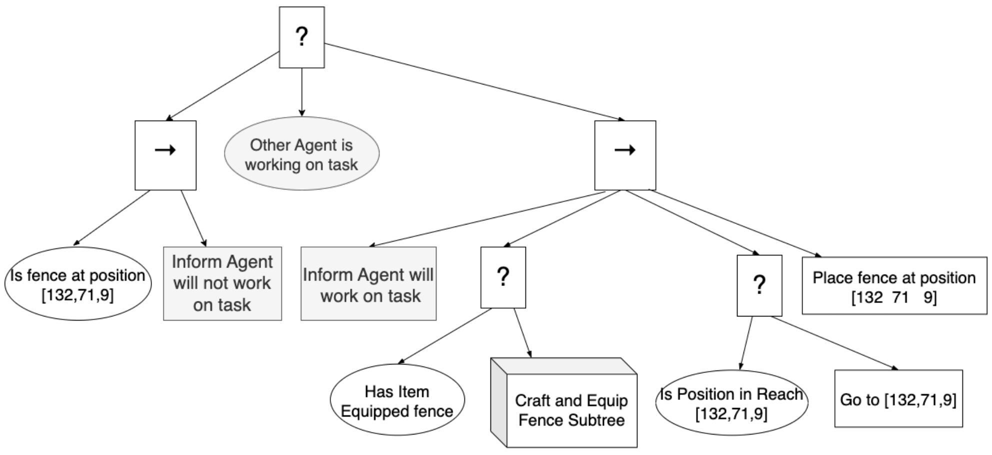
Figure 6.1a shows the PPA-BT for G_0, while Figure 6.1b shows the modified collaborative BT for G_0.

6.1.3 Reactive cooperation

To estimate the agents' ability to handle sudden hazards and investigate the benefit of conditions that let them consider other agents than themselves, they were evaluated in a modified fence post experiment where a hostile enemy could attack the agents. The scenario was altered by adding an enemy in the


?
Is fence at position [132 71 9]
→
?
?
Place fence at position [132 71 9]
Has Item Equipped fence
Craft and Equip Fence Subtree
Is Position Within Reach [132 71 9]
Go to [132 71 9]

(a) Traditional backward chaining


?
→
Other Agent is working on task
→
Is fence at position [132,71,9]
Inform Agent will not work on task
Inform Agent will work on task
?
?
Place fence at position [132 71 9]
Has Item Equipped fence
Craft and Equip Fence Subtree
Is Position in Reach [132,71,9]
Go to [132,71,9]

(b) Collaborative backward chaining.

Figure 6.1: Traditional backward chaining (Algorithm 1) compared to collaborative backward chaining (Algorithm 2), for the postcondition $C_{\text{post}} =$ **Is Block at Position** $\vec{p}$. The gray conditions are the receiver nodes, and the gray actions are the sender nodes.

environment and adding a condition that no hostile enemies are nearby. To assess the advantage of observing other agents, the condition of no hostiles nearby was varied to consider either the enemies near only the agent itself or the enemies near both the agent itself and other agents in the agent's observation state.

The modified scenario transpired in a world that the FWG generated. While similar results could be obtained in the DWG, the absence of altitude differences lowered the requirements on the agents' combat skills. The actions to defeat an enemy were advanced enough for two agents to defeat the aggressor consistently, but advanced combat AI was not in the thesis scope.

The two changes in the world generation were adding an enemy, a zombie equipped with a stone sword. The zombie adds another hyperparameter, $\vec{p}_e$, the initial position of the enemy. The experiments used a zombie since zombies have a strong defense but relatively weak attacks. Thus, zombies are too strong for a single unarmed agent to defeat but not too strong for two. Each agent was equipped to lower the damage from the zombie giving the supporting agent enough time to reach and save the assailed agent. A lone agent with no weapons was strong enough to sustain until help arrived, but too weak to defeat the zombie.

In the reactive cooperation experiments, the backward chaining algorithm used the goals

$$\vec{G}' = G_r \cup \vec{G}, \tag{6.4}$$

where G_r is a condition that no hostile enemies should be near the agent and $\vec{G}$, as defined in equation (6.1), are the goals to place the fence posts in the grid pattern. Note that G_r must be placed at the beginning of the set of goals (before the fence post goals). Since the root of the backward chained BT is a sequence performing the goals by their order in $\vec{G}'$, the agent prioritizes defeating hostiles above placing fence posts.

Two different variations of G_r were evaluated,

$$G_r^1 = \textbf{Is no enemy nearby}, \tag{6.5}$$

which only considers the agent's near surroundings and

$$G_r^2 = \textbf{Is no enemy near any agent}, \tag{6.6}$$

which observed other agents and considered whether there were enemies close to them. The PPA triplets associated with these two goals and other PPA triplets required in this experiment are presented in Table 6.2.

The reactive cooperation experiments used the following hyperparameters

- n_t - number of trials for each combination of n_a and $\mathcal{S}$.

	C_i^{post}	C_i^{pre}	A_i
PPA_9	Is no enemy near	Is enemy in reach	Attack enemy
PPA_{10}	Is enemy in reach	None	Go to enemy
PPA_{11}	Is no enemy near any agent	Is enemy near agent in reach	Attack enemy near agent
PPA_{12}	Is enemy near agent in reach	None	Go to enemy near agent

Table 6.2: PPA triplets used for the reactive cooperation tests in the fence post placement scenario. These triplets were used for the reactive cooperation tests in the fence post placement in addition to the triplets in Table 6.1.

- n_a - number of agents.
- $\mathcal{S}$ - the system used to control the agents; collaboration with backup (Algorithm 3), collaboration without backup (Algorithm 2), or traditional backward chaining (Algorithm 1).
- $\mathbf{P}_a = (\vec{p}_a^{\,i})_i^{n_{agents}}$ - The agents' initial positions.
- $\mathcal{I}$ - The initial inventory of the agents.
- Δ - the vertical and horizontal distance between nearby fence posts.
- $\vec{p}_0 = (x_0, y_0, z_0)$ - the position of the fence post center.
- G_r - The goal that was used. Either G_r^1, as defined in Equation (6.5), or G_r^2, as defined in Equation (6.6).
- $\vec{p}_e$ - The initial position of the enemy.
- s - The randomization seed used by the FWG to generate the world.
- G - The generator configuration string.

Adapting the condition to a MAS slightly breaks the requirement not to include any editing of PPAs or behaviors. However, there are two justifications for still doing these experiments. First, this alteration is not a part of the suggested method but a further investigation that occurred in the scope of the thesis. Second, the collaborative condition is designed to be an improvement regardless of the BT has been adapted to handle cooperation beforehand. In this case, it is dealt with by not setting the goals that benefit from multiple agents working together as distributable.

6.2 Stone pickaxe materials gathering

The second scenario had two or more agents working together to gather resources used for crafting stone pickaxes. Crafting a stone pickaxe requires both stone and wood, where the processes of gathering stone and wood are significantly different. Therefore the optimal approach is to have one agent gather wood and the other gather stone. This scenario was selected to highlight the flaws of the naive collaborative extension (Algorithm 2), highlighting the advantage of adding backup as in the refined proposal (Algorithm 3) to prevent idling agents. This scenario evaluated the agents in two different settings; an authentic world procedurally generated by the DWG and a manually crafted test arena constructed to visualize the impact of goal-switching cost.

6.2.1 Overview

The stone pickaxe experiment had two scenario-specific hyperparameters. The first is the starting inventory $\mathcal{I}$. The second one is the number of stone pickaxes n_p that the agents should gather materials to craft. n_p has to be large enough for the result to be independent of the experiment setup. And small enough for a reasonable completion time.

While agents theoretically can distribute items between them, Project Malmo does not support item distribution. Therefore, the agents do not craft the stone pickaxes in the experiment. Instead, the goal was to gather the necessary materials for constructing the stone pickaxes.

The modified backward chaining algorithm was executed on a set of goals $\vec{G} = [G_1, G_2]$ where

$$G_1 = \textbf{Agents have 2n}_\textbf{p}\textbf{ sticks}\text{ and} \tag{6.7}$$
$$G_2 = \textbf{Agents have 3n}_\textbf{p}\textbf{ stone blocks} \tag{6.8}$$
$$. \tag{6.9}$$

Note that G_1 is not identical to

$$G' = \textbf{Has 2n}_\textbf{p}\textbf{ sticks}, \tag{6.10}$$

a modification of the post-condition of PPA$_5$ and PPA$_6$ in Table 6.1. While G' only counts items in the agent's own inventory, G_1 counts the system's combined item pool. Both goals in $\vec{G}$ are distributable in the stone pickaxe scenario. Then, when two agents used the collaborative modification, one

	C_i^{post}	C_i^{pre}	A_i
PPA$_1$	Agents have n I (craftable)	[Has item m_i]$m_i{}^{M(I)}$	Craft item I
PPA$_2$	Agents have n I (gatherable)	Has item $G(I)$ equipped, Item I as pickup near	Craft item I
PPA$_3$	Has item I equipped	Has item I	Equip item I
PPA$_4$	Has item I (craftable)	[Has item m_i]$m_i{}^{M(I)}$	Craft item I
PPA$_5$	has item I (gatherable)	Has item $G(I)$ equipped, Item I as pickup near	Pick up item I
PPA$_6$	Item I as pickup near	block $B(I)$ is in reach	Break block $B(I)$
PPA$_7$	Block B is in reach	Block B is found	Go to block B
PPA$_8$	Block B is observable	None	Look for block

Table 6.3: PPA triplets used for the stone pickaxe materials scenario. The backward chaining algorithm used the triplets to create the PPA-BT for solving the scenarios. Here B is a block, I, an item, $\vec{p}_i$, a 3d position, $B(I)$, the block form of I, $M(I)$, the material needed for crafting I, and $G(I)$, the tool needed for gathering I.

agent worked on achieving G_1 and the other on achieving G_2.

To know the total amount of items among all the agents, G_1 used the blackboard. The blackboard had a channel CH_O^a for each agent and item type. The $B(CH_O^a)$ value is updated when an agent gathers or crafts an item.

As shown in Table 6.3, The gathering experiment reused most of the PPA triplets from the fence post experiment, presented in Table 6.1 and Table 6.2. The significant difference is an additional PPA triplet associated with the postcondition **Agents have n I**, where I is an item type, and n is an amount. Figure A.2 contains the PPA-tree generated by the default backward chaining algorithm (Algorithm 1).

6.2.2 Highlighting flaws of the naive collaborative extension

The stone pickaxe scenario was formed to highlight several issues with the naive collaborative method depicted in Algorithm 2. The four differences between the scenarios showcasing the flaws are:

- The time loss of switching between goal G_i to goal G_j, l_j^i is smaller. The stone and wood sources are relatively close, while the fence posts are further away.

- In the stone pickaxe scenario, the goals are completed faster if several agents work together on completing the same goal. In the fence post scenario, on the other hand, two agents trying to achieve the same goal are detrimental.
- The number of goals, $n_G = \text{len}(\vec{G})$ is smaller in the stone pickaxe scenario.
- There is a larger difference in completion time between the goals.

The stone pickaxe scenario only has two goals, G_1 and G_2. Assuming that G_1 takes t_1 for a single agent to complete, G_2 takes t_2 for a single agent to complete and $t_1 \ll t_2$. The goals are independent. Achieving G_1 does not decrease the completion time of G_2. The issue was even more noteworthy because of the small number of goals. A system $\mathcal{S}_I$ with classic backward chained BTs (generated by Algorithm 1) completes both goals in

$$T_G^I = (t_1 + t_2)/2 + l_{12} \approx t_2/2 + l_{12}. \tag{6.11}$$

On the other hand, a system $\mathcal{S}_C$ with collaborative BT (generated by Algorithm Algorithm 2) achieves both goals in

$$T_G^C = \text{max}(t_1, t_2) = t_2 \tag{6.12}$$

Therefore, reserving tasks is optimal as long as $l_{12} > t_2/2$. On the other hand, if $l_{12} < t_2/2$, a traditional approach works the best. Nevertheless, a system $\mathcal{S}_{CC}$, with collaboration with backup, with BTs generated by Algorithm 3 outperforms either approach.

Test arena

The stone pickaxe scenario evaluated the agents in two different settings; a procedurally generated world and a manually composed test arena. The test arena has a chunk of wood and a chunk of stone on opposite sides of agents' start positions, in contrast to the DWG scattering stone and wood blocks randomly in the environment. The design of the test arena emphasizes the effect of the goal-switching cost. A considerable distance between the stone and the wood increased the goal-switching cost. Each agent starting with a wooden pickaxe also increased the goal-switching cost.

The intersection of the stone and wood densities in the authentic Minecraft world minimized the cost of switching goals. Albeit stones typically exist at

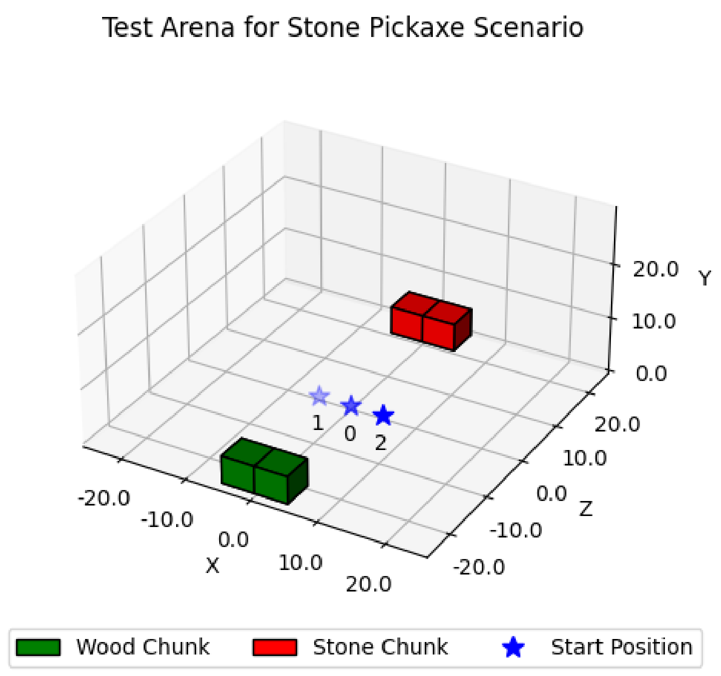


Figure 6.2: An abstract representation of the test arena in the XZ plane.

a lower altitude than wood, the difference is neither absolute nor significant. The low goal-switching cost warranted a test arena with separated wood and stone sources. The test arena was square, with the agents' start positions in the center and the wood and stone chunks at opposing edges. The substantial distance between the wood and the stone chunks significantly increased the goal-switching cost.

The test arena utilizes the FWG, but contrarily to the fence post scenario, the generator places no trees or other decorators. Since the test arena use the FWG, it also requires the generator configuration string G. The chunks are cuboids, defined by coordinates of a pair of opposite corners, $\mathbf{X} = [(x_0, y_0, z_0), (x_1, y_1, z_1)]$. The coordinate pairs for the stone chunk are $\mathbf{X}_S$, and the coordinates for the wood chunk are $\mathbf{X}_W$.

Figure 6.2 depicts a visual representation of the test environment on the XZ plane.

Additionally, since gathering stone requires a wooden pickaxe, each agent started with one. If the stone gatherer had needed to craft the wooden pickaxe itself, it would have to go to the wood chunk to collect the wood before it could start gathering stone. Requiring the stone gatherer to go the wood chunk completely redacts the potential goal-switching cost of gathering wood for crafting the sticks. Therefore, the agents start with all prerequisites to strictly divide the tasks, setting the conditions for proving that the collaborative approach counteracts the considerable goal-switching cost.

6.2.3 Hyperparameters

The stone pickaxe crafting experiment had slightly different hyperparameters in the authentic world generated by the DWG and the test arena. The test arena had three more hyperparameters than the authentic world. The first hyperparameter is the generator string G, required since the test arena used the FWG. The second and third map-specific hyperparameters are $\mathbf{X}_W$, the coordinates of the wood chunk, and $\mathbf{X}_S$, the coordinates of the stone chunk.

The hyperparameters used in the authentic world experiments were

- n_t - number of trials for each combination of n_a and $\mathcal{S}$.
- n_a - number of agents.
- $\mathcal{S}$ - the system used to control the agents; collaboration with backup (Algorithm 3), collaboration without backup (Algorithm 2), or traditional backward chaining (Algorithm 1).
- $\mathbf{P}_a = (\vec{p_a^i})_i^{n_{agents}}$ - The initial positions of the agents.
- $\mathcal{I}$ - The initial inventory of the agents.
- n_p - the number of stone pickaxes to gather materials for.
- s - The randomization seed used by the DWG to generate the world.

The hyperparameters used in the test arena were

- n_t - number of trials for each combination of n_a and $\mathcal{S}$.
- n_a - number of agents.
- $\mathcal{S}$ - the system used to control the agents; collaboration with backup (Algorithm 3), collaboration without backup (Algorithm 2), or traditional backward chaining (Algorithm 1).
- $\mathbf{P}_a = (\vec{p_a^i})_i^{n_{agents}}$ - The initial positions of the agents.
- $\mathcal{I}$ - The initial inventory of the agents.

- n_p - the number of stone pickaxes to gather materials for.
- s - The randomization seed for generating the world.
- G - The generator configuration string.

Chapter 7

Results and discussion

The research question of this thesis, see Section 1.2, was

> ”How can backward chained BTs provide a good solution for multi-agent coordination in complex adversarial environments?”

As will be seen in this Chapter, one good solution is agents that observe and assess the threat levels of other agents in the MAS (as proposed in Chapter 4). Another good solution that does not require tailormade PPAs is 1-to-1 task distribution (as proposed in Chapter 5).

The results of the two scenarios verified four essential hypotheses from Chapter 4 and Chapter 5.

- MASs where the agents observe other agents and help them when in danger has a higher chance of success than agents which do not consider the state of other agents in the MAS.
- The naive collaborative extension (Algorithm 2) performs better than traditional backward chaining (Algorithm 1) in the fence post scenario.
- The naive collaborative extension (Algorithm 2) performs worse than traditional backward chaining (Algorithm 1) in the stone pickaxe scenario in the generated world but not in the test arena.
- The proposed refined approach (Algorithm 3) delivers the best outcome in both scenarios.

The methods were evaluated by their average completion time for various amounts of agents n_a. Additionally, the alteration did not have any significant impact on single-agent systems.

The chapter has two sections. Section 7.1 presents the results of each scenario. Afterward, Section 7.2 discusses the implications of the results.

7.1 Results

The section contains results from both scenarios. Section 7.1.1 shows the results from the fence post placement, and Section 7.1.2 shows the results from the stone pickaxe scenario. The most important results for both scenarios were the time the agents with differently generated BTs took to complete the goals. There were two primary variable hyperparameters, the number of agents, n_a, and the system $\mathcal{S}$. $\mathcal{S}$ was one of three different systems:

- $\mathcal{S}_I$ - A MAS with n_a agents with identical BTs, $\mathcal{T}_0$, generated by the traditional backward chaining algorithm defined in Algorithm 1.
- $\mathcal{S}_C$ - A MAS with n_a agents with identical BTs[1], $\mathcal{T}_0'$, generated by the collaboratively extended backward chaining algorithm depicted in Algorithm 2.
- $\mathcal{S}_{CC}$ - A MAS with n_a agents with identical BTs [1], $\mathcal{T}_B$, defined by Algorithm 3.

The scenarios were compared by their efficiency, measured by the average completion time for the system to achieve all the goals, $\vec{G}$. All experiments had a maximum time limit of $T = 300$ seconds.

7.1.1 Fence post placement

The experiments ran in two different worlds in the fence post placement scenario. An authentic one generated by the Minecraft DWG and a flat world generated by the custom FWG [43]. Cooperative backward chaining, with and without backup, was evaluated in both worlds. However, two additional experiments were assessed in the flat world generated by the FWG. The first of the additional experiments investigate how Δ, the distance between fence posts, affects the performance of the different systems. The second additional experiment investigates the importance of considering other agents when the agents are subjected to a hostile entity.

Collaborative backward chaining in an authentic world

Figure 7.1 displays the average completion times in an authentic world generated by the DWG. Further, Table 7.1 shows the hyperparameters specific to the experiments evaluated in the authentic world.

[1] The only difference between the BTs was limited to the agent identification signature N_k used for task reservation.

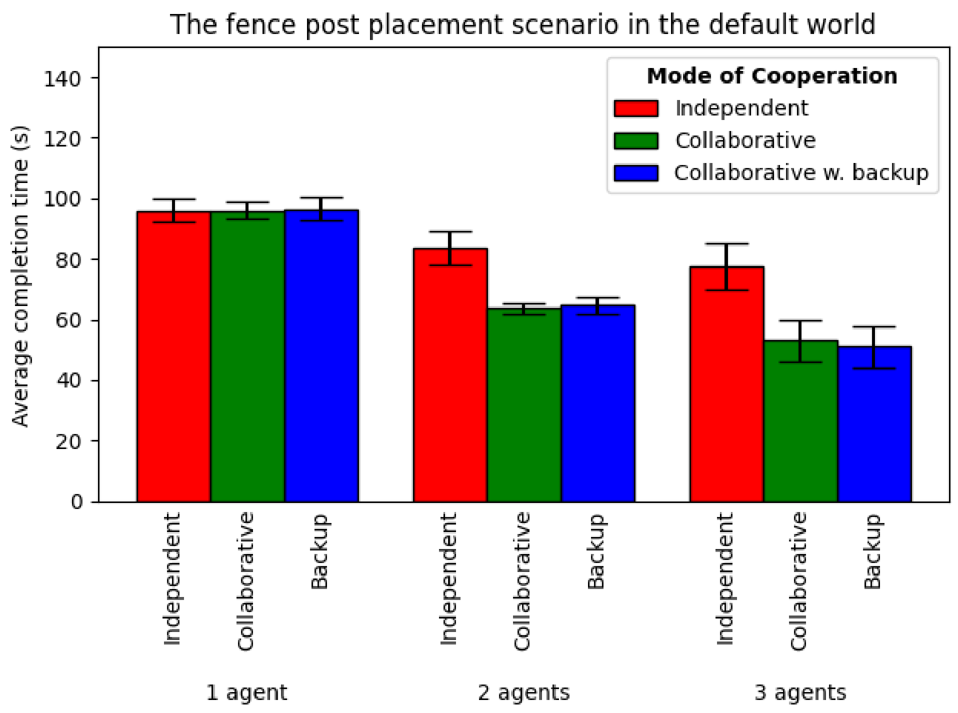


Figure 7.1: Average completion time for the fence post placement scenario in the authentic Minecraft world generated by the DWG. For each number of agents, there are three bars; the left shows the completion time of the independent baseline design (Independent, Algorithm 1), the middle shows the completion time of the simple unrefined collaborative design (Collaborative, Algorithm 2), and the right shows the completion time of the proposed collaborative design (Backup, Algorithm 3). The standard deviation is indicated for each bar. As can be seen, with 2 and 3 agents present, collaboration is useful when placing fences.

n_t	45
n_a	$[1, 2, 3]$
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[101, 71, 9], [132, 71, -21], [162, 71, 9]]$
Δ	7
$\vec{p}_0$	$[132, 7, 9]$
s	4000020

Table 7.1: Hyperparameters used in the fence post placement scenario in the authentic world generated by the DWG. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, Δ, the horizontal and vertical distance between grid points, $\vec{p}_0$, the center coordinate of the grid, and s the generation seed.

n_t	15
n_a	$[1, 2, 3]$
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[101, 9, 9], [132, 9, -21], [162, 9, 9]]$
Δ	15
$\vec{p}_0$	$[132, 9, 9]$
s	4000020
G	*3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;35;decoration*

Table 7.2: Hyperparameters used in the fence post placement scenario in the flat world generated by the FWG. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, Δ, the horizontal and vertical distance between grid points, $\vec{p}_0$, the center coordinate of the grid, s, the generation seed and G the flat world generator string.

Collaborative backward chaining in a flat world

Figure 7.2 shows the average completion times for the world generated by the FWG. Meanwhile, Table 7.2 shows the hyperparameters associated with the sub-scenario. Additionally, Figure 7.3 depicts the paths of the agents in the flat world for $n_a = 3$.

Collaborative backward chaining with variable distance between fence posts

The fence post placement scenario also evaluated the agents for various values of Δ in the flat world, depicted in Figure 7.4 and Figure 7.5. The figure shows two different measure points. The completion times for the systems and the

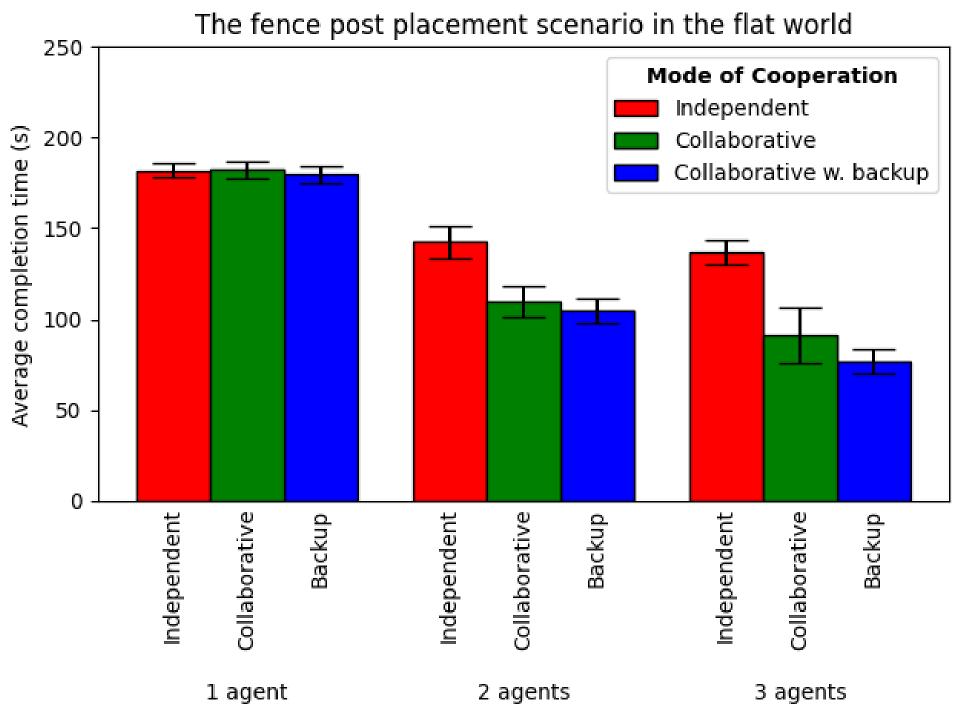


Figure 7.2: Average completion time for the fence post placement scenario in the flat world generated by the FWG. For each number of agents, there are three bars; the left shows the completion time of the independent baseline design (Independent, Algorithm 1), the middle shows the completion time of the simple unrefined collaborative design (Collaborative, Algorithm 2), and the right shows the completion time of the proposed collaborative design (Backup, Algorithm 3). The standard deviation is indicated for each bar. As can be seen, collaboration is useful when placing fences with 2 and 3 agents present.

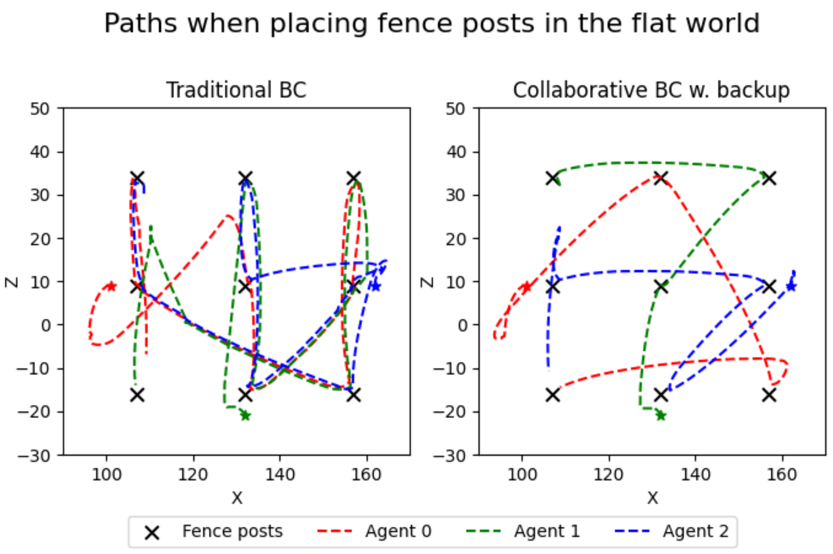


Figure 7.3: The paths of the agents in a single experiment trial from the fence post-placement scenario in the flat world. The experiment uses $n_a = 3$ and the hyperparameters presented in Table 7.2. The two figures use different BTs. The left figure uses traditional backward chaining as presented in Algorithm 1, and the right figure uses the collaborative backward chaining approach as presented in Algorithm 3. In each figure, the dashed lines represent the paths each agent took. The stars denote the starting point of the agents. Finally, the crosses symbolize the fence post target positions from the goal set. Note that the trees where the agent gathers material are not indicated in the figures but still give rise to sudden changes in trajectory direction. Also note that in the non-collaborative approach, several fence positions are approached by several agents, as they are not aware that other agents are working on solving the same tasks as themselves.

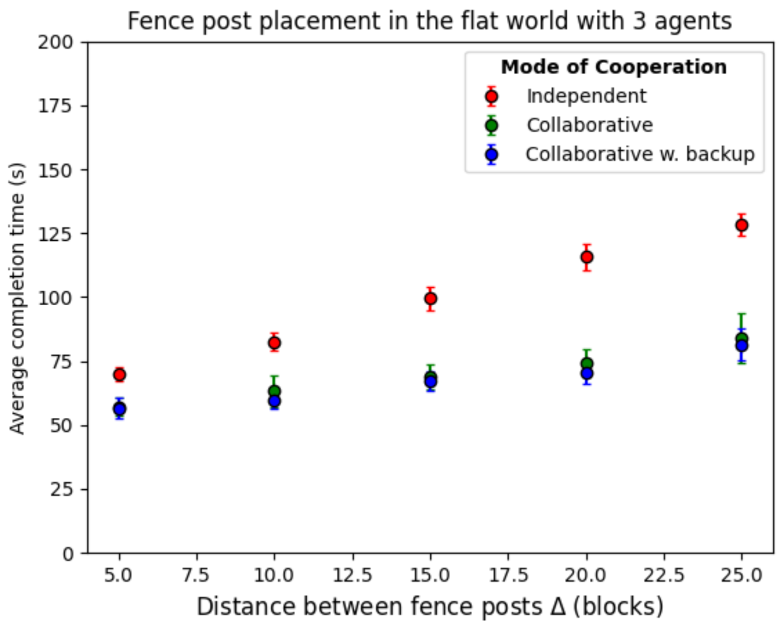


Figure 7.4: Average completion time in the fence post placement scenario in the flat world for different values of Δ. The figure uses $n_t = 15$ and $n_a = 3$. The points show the average completion time for the different methods using different distances between the fence posts. The blue dots show the average time for the traditional backward chained algorithm as shown in Algorithm 1. The red dots show the average time for collaborative backward chaining without backup as presented in Algorithm 2. The green dots show the average time for collaborative backward chaining without backup as presented in Algorithm 3. The figures also include error bars denoting the standard deviation.

fraction of the completion time of $\mathcal{S}_C$ and $\mathcal{S}_{CC}$ relative to that of $\mathcal{S}_I$. The experiment ran for $n_t = 15$ trials for each combination of hyperparameter, system, n_a, and Δ. Table 7.3 lists the associated hyperparameters

Reactive cooperation

Finally, the fence post placement scenario also evaluated the advantage of considering other agents in the context of a hostile entity being placed in the arena. The success rate of completing the fence post placement in the flat world with an enemy is shown in Figure 7.6. The isolated agent used a goal $G_r^1 =$ **Is close to enemy**, and the observant agent used a goal $G_r^2 =$ **Is any agent close to enemy**. Figure 7.7 shows the average time for the

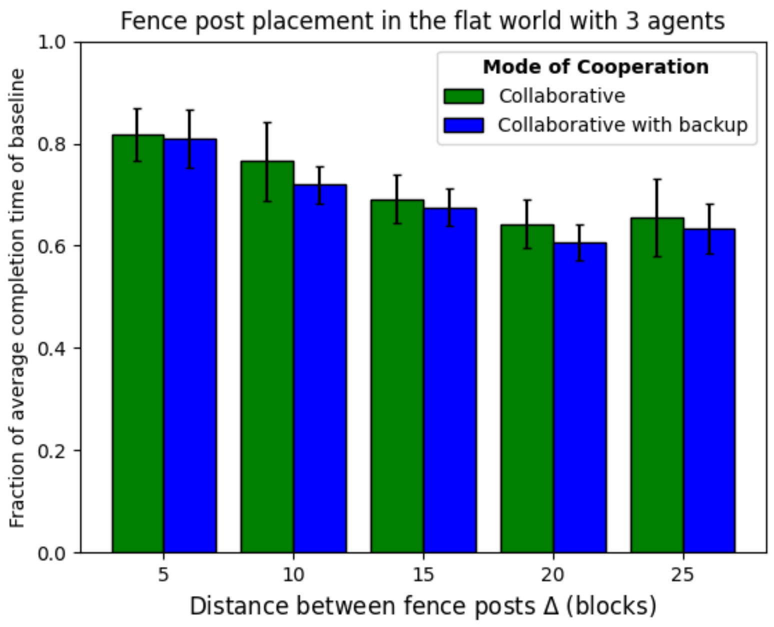


Figure 7.5: Average completion time fraction relative to baseline in the fence post placement scenario in the flat world for different values of Δ. The figure uses $n_t = 15$ and $n_a = 3$. The bars show the fraction of the completion time for the different methods relative to the baseline, the traditional backward chaining. The figures also include error bars denoting the standard deviation.

n_t	15
n_a	3
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[101, 9, 9], [132, 9, -21], [162, 9, 9]]$
s	4000020
Δ	$[5, 10, 15, 20, 25]$
$\vec{p}_0$	$[132, 9, 9]$
G	*3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;35;decoration*

Table 7.3: Hyperparameters used for variable fence post distance experiments in the fence post placement scenario in the flat world. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, Δ, the horizontal and vertical distance between grid points, $\vec{p}_0$, the center coordinate of the grid, s, the generation seed and G the flat world generator string.

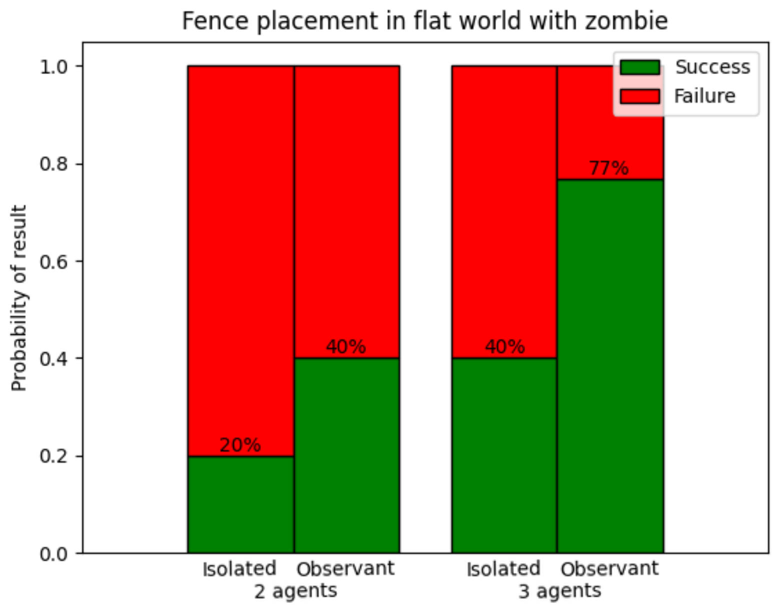


Figure 7.6: Success rate in the placement scenario in the flat world with a hostile entity, a zombie equipped with a stone sword, with observant or isolated agents. The isolated agent used a goal G_r^1 = **Is no enemy nearby**, and the observant agent used a goal G_r^2 = **Is no enemy near any agent**. The bars show the success rate. A failure means that the zombie defeated all agents.

reactive cooperation experiment. The evaluation of reactive cooperation used collaboration with backup. Table 7.4 shows the hyperparameters used in the experiments.

7.1.2 Stone pickaxe materials gathering

As the fence post placement scenario, the gathering scenario used two worlds. One test environment was generated by the Minecraft DWG, and the other was manually designed to induce significant goal-switching costs. The goal-switching cost was maximized by distancing wood and stone sources, as well as the agents' start positions. In the manually designed test arena, the agents could start with a pickaxe. The agents were evaluated with different pickaxes.

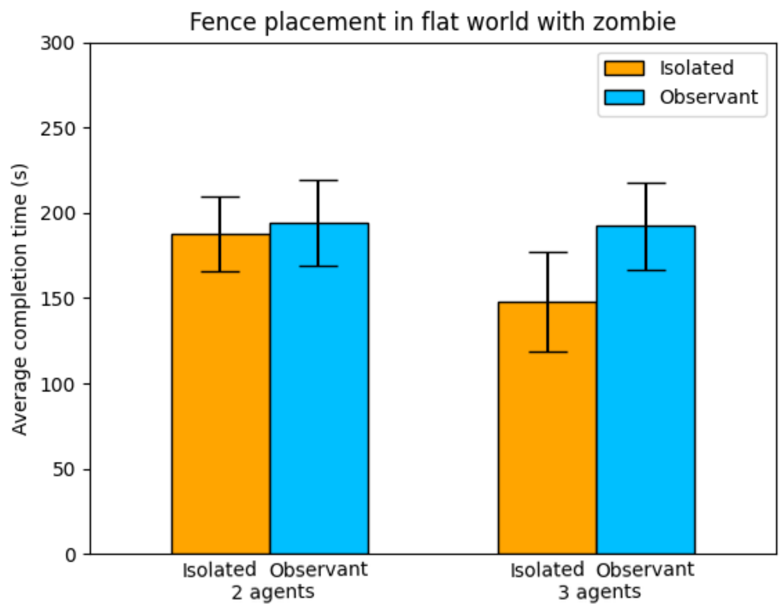


Figure 7.7: Fraction of completion times in the placement scenario in the flat world with a hostile entity, a zombie equipped with a stone sword, with observant or isolated agents. The isolated agent used a goal G_r^1 = **Is no enemy nearby**, and the observant agent used a goal G_r^2 = **Is no enemy near any agent**. The points show the fraction of the completion time for the different methods relative to the baseline, the traditional backward chaining. The figures also include error bars denoting the standard deviation.

n_t	30
n_a	$[2, 3]$
$\mathcal{S}$	$\mathcal{S}_{CC}$
$\mathbf{P}_a$	$[[101, 9, 9], [132, 9, -21], [162, 9, 9]]$
Δ	15
$\vec{p}_0$	$[132, 9, 9]$
G_r	$[G_r^1, G_r^2]$
$\vec{p}_e$	$[116, 10, 9]$
s	4000020
G	*3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;35;decoration*

Table 7.4: Hyperparameters used for the reactive cooperation experiments in the fence post placement scenario in the flat world. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, $\mathcal{I}$, the agents' starting inventory, Δ, the horizontal and vertical distance between grid points, $\vec{p}_0$, the center coordinate of the grid, G_r, the enemy avoidance goal, $\vec{p}_e$, the enemy starting position, s, the generation seed, and G, the flat world generator string,

Collaborative backward chaining in an authentic world

Figure 7.8 portrays the completion times for the stone pickaxe scenario in the authentic world generated by the DWG. Table 7.5 shows the other hyperparameters.

Collaborative backward chaining in a test arena

Figure 7.9 shows the completion times for an agent equipped with a wooden pickaxe in the test arena. The experiment ran for $n_t = 15$ trials. Table 7.6 lists the other hyperparameters. Figure 7.10 shows the paths of the agents in the test arena for $n_a = 2$.

Collaborative backward chaining in a test arena with variable starting inventory

Additionally, we evaluated the impact of the agents' starting inventories. All agents started the experiments with the same items. The inventory could either be empty or contain a single pickaxe made of wood, stone, iron, or diamond. Note that the agents starting with stone pickaxes did not impact the amount of stone and wood they had to gather. The experiments ran in both experiment environments, the test arena and the authentic world generated by the DWG.

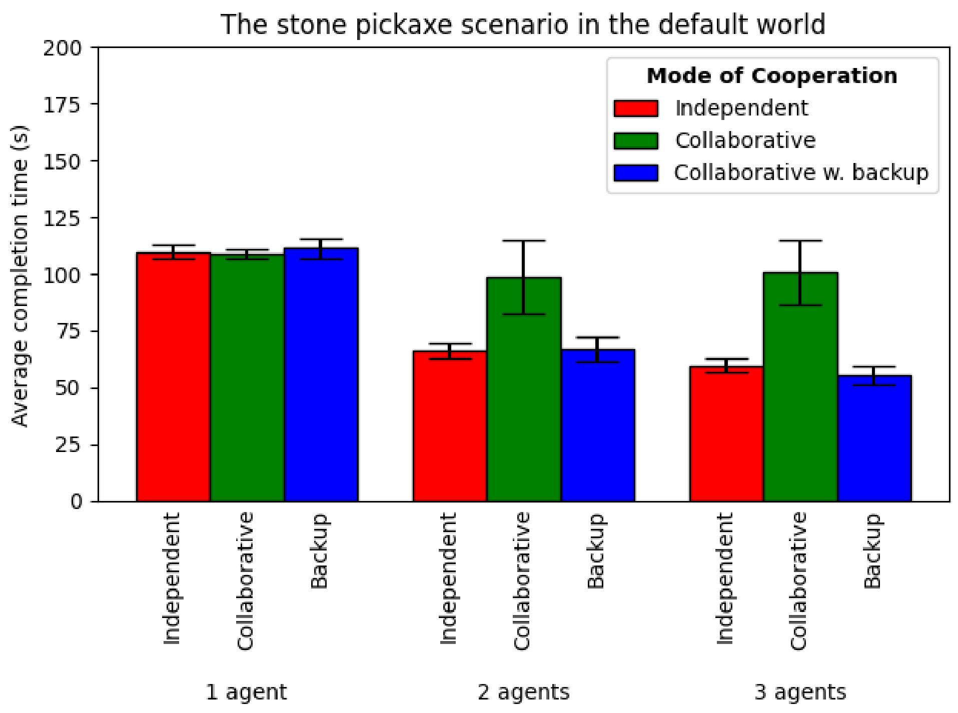


Figure 7.8: Average completion time for the stone pickaxe scenario in an authentic world generated by the Minecraft DWG. For each number of agents, there are three bars; the left shows the completion time of the independent baseline design (Independent, Algorithm 1), the middle shows the completion time of the simple unrefined collaborative design (Collaborative, Algorithm 2), and the right shows the completion time of the proposed collaborative design (Backup, Algorithm 3). The standard deviation is indicated for each bar.

n_t	15
n_a	$[1, 2, 3]$
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[131, 72, 17], [132, 72, 4], [140, 72, 24]]$
$\mathcal{I}$	None
n_p	5
s	4000020

Table 7.5: Hyperparameters used by the stone pickaxe crafting scenario in the authentic world generated by the DWG. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, $\mathcal{I}$, the starting inventory, n_p, the number of pickaxes to gather materials for, and s, the generation seed.

n_t	15
n_a	3
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[0, 9, 0], [-5, 9, 0], [5, 9, 0]]$
$\mathcal{I}$	A wooden pickaxe
n_p	5
s	4000020
G	*3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;35*

Table 7.6: Hyperparameters used by the stone pickaxe scenario in the test arena. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, $\mathcal{I}$, the starting inventory, n_p, the number of pickaxes to gather materials for, s, the generation seed, and G, the generator configuration string.

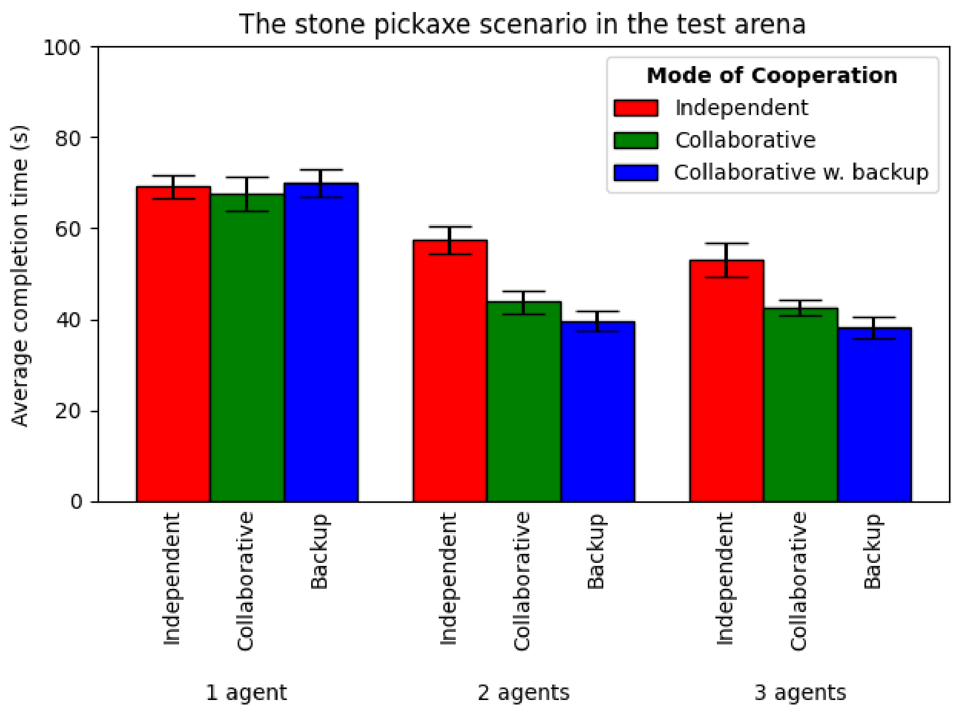


Figure 7.9: Average completion time for the stone pickaxe scenario in the test arena. For each number of agents, there are three bars; the left shows the completion time of the independent baseline design (Independent, Algorithm 1), the middle shows the completion time of the simple unrefined collaborative design (Collaborative, Algorithm 2), and the right shows the completion time of the proposed collaborative design (Backup, Algorithm 3). The standard deviation is indicated for each bar.

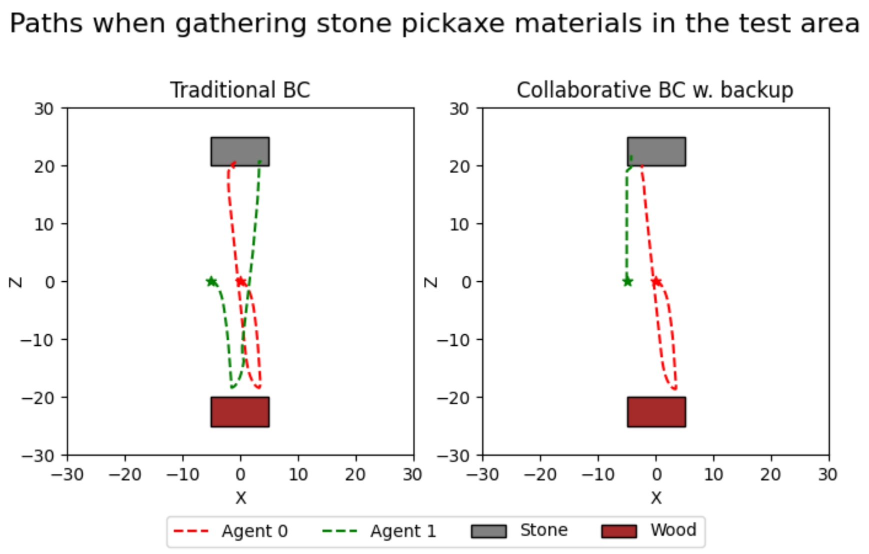


Figure 7.10: The paths of the agents in a single experiment trial from the stone pickaxe materials gathering scenario in the test arena. The experiment uses $n_a = 2$ and the hyperparameters presented in Table 7.6. The two figures use different methods. The left figure uses the classic backward chaining shown in Algorithm 1, and the right figure uses collaborative backward chaining with backup as presented in Algorithm 3. In each figure, the dashed lines represent the paths each agent took. The stars represent the starting point of the agents. The rectangles denote material chunks made by blocks of a single material. Each chunk is three blocks tall, six blocks deep, and eleven blocks wide. Besides the depicted chunks, the entire test arena is a void flat world without other generated or manually placed blocks. Note that in the left figure, all agents start moving toward the wood, whereas in the right figure, one agent goes to pick up the stone, and two go to pick up the wood.

n_t	15
n_a	$[1, 2, 3]$
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[0, 9, 0], [-5, 9, 0], [5, 9, 0]]$
$\mathcal{I}$	$\mathcal{I}_P$
n_p	5
s	4000020
G	*3;1*minecraft:bedrock,7*minecraft:dirt,1*minecraft:grass;35*

Table 7.7: Hyperparameters used by the stone pickaxe scenario with variable starting inventory in the test arena. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, $\mathcal{I}$, the starting inventory, n_p, the number of pickaxes to gather materials for, s, the generation seed, and G, the generator configuration string. Here $\mathcal{I}_P$ =[None, A wooden pickaxe, A stone pickaxe, An iron pickaxe, A diamond pickaxe] is the set of pickaxes.

First, Figure 7.11 and Figure 7.12 show the completion times of the agents in the test arena with a variable starting inventory $\mathcal{I}$. Second, Figure 7.13 and Figure 7.14 show the completion times of the agents in the authentic world with a variable initial inventory. For the variable inventory experiments, Table 7.7 shows the hyperparameters used in the test arena, while Table 7.8 shows the hyperparameters used in the authentic world.

7.2 Discussion

The results motivate two distinct meaningful discussion topics. The advantage of reactive cooperation and the benefits of 1-to-1 task distribution in backward chaining.

7.2.1 Reactive cooperation

MASs with agents that observe and evaluate threats towards other agents in the system have a higher chance of success than agents which only consider their own danger risks. It is straightforward to create BTs to control agents to assess threats toward any agent in any-sized MASs. This is demonstrated in the experiments with a hostile entity, where a zombie equipped with a stone sword, appears amid the agents and tries to stop them from completing their objectives.

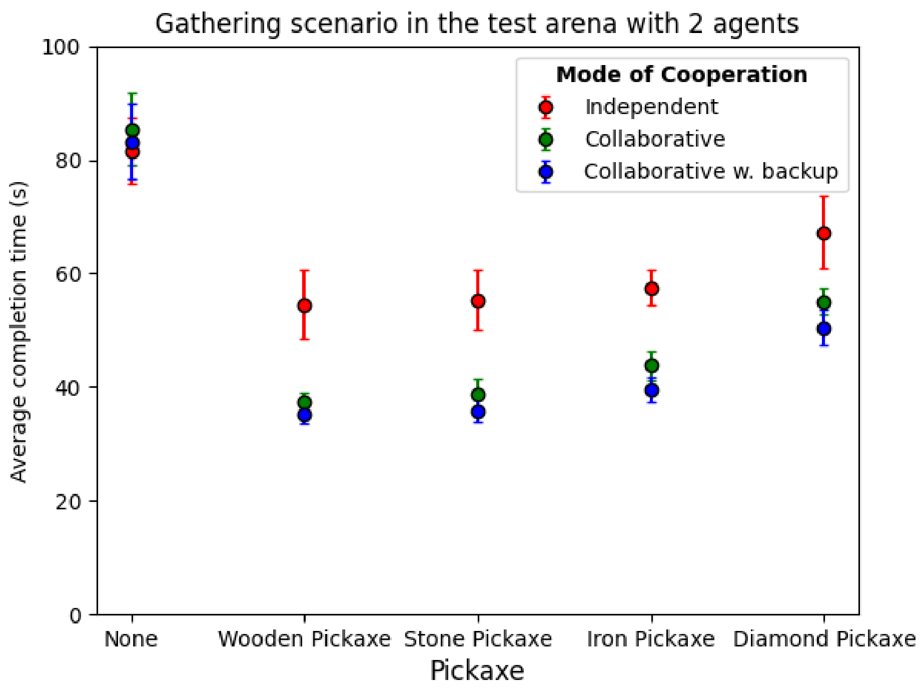


Figure 7.11: Average completion time for the stone pickaxe scenario in the test arena for two agents starting with different inventories. Each figure depicts the average completion time from 15 trials of each parameter combination; system, starting inventory, and the number of agents. The agents could either have no items or start with a single pickaxe. The material of the pickaxe was either wood, stone, iron, or diamond. The figures show the results for systems with $n_a = 2$. Showing the results for a single agent system, with $n_a = 1$, is redundant as the difference between the systems was trivial. All agents in the system started with the same inventory. The points show the average completion time for the different methods. The blue dots show the average time for the traditional backward chained algorithm as shown in Algorithm 1. The red dots show the average time for collaborative backward chaining without backup as presented in Algorithm 2. The green dots show the average time for collaborative backward chaining without backup as presented in Algorithm 3. The figures also include error bars denoting the standard deviation.

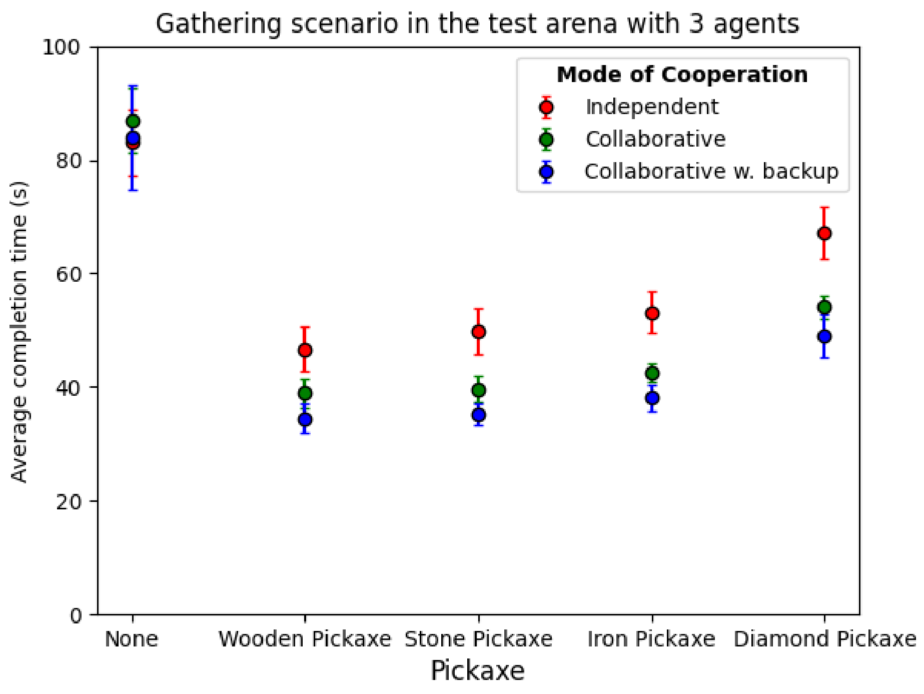


Figure 7.12: Average completion time for the stone pickaxe scenario in the test arena for three agents starting with different inventories. Each figure depicts the average completion time from 15 trials of each parameter combination; system, starting inventory, and the number of agents. The agents could either have no items or start with a single pickaxe. The material of the pickaxe was either wood, stone, iron, or diamond. The figures show the results for systems with $n_a = 3$. Showing the results for a single agent system, with $n_a = 1$, is redundant as the difference between the systems was trivial. All agents in the system started with the same inventory. The points show the average completion time for the different methods. The blue dots show the average time for the traditional backward chained algorithm as shown in Algorithm 1. The red dots show the average time for collaborative backward chaining without backup as presented in Algorithm 2. The green dots show the average time for collaborative backward chaining without backup as presented in Algorithm 3. The figures also include error bars denoting the standard deviation.

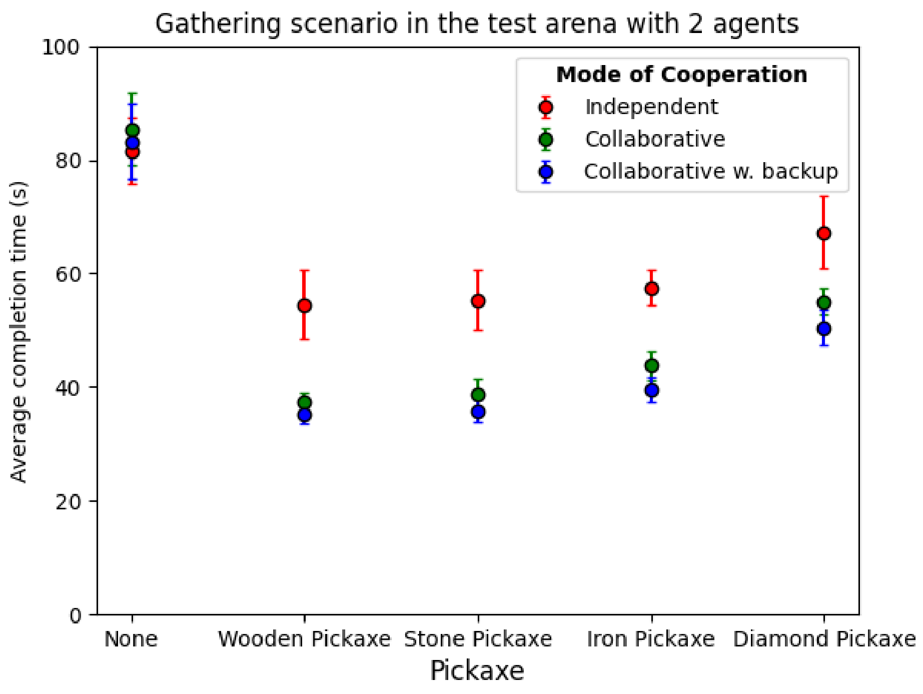


Figure 7.13: Average completion time for the stone pickaxe scenario in the authentic world for two agents starting with different inventories. Each figure depicts the average completion time from 15 trials of each parameter combination; system, starting inventory, and the number of agents. The agents could either have no items or start with a single pickaxe. The material of the pickaxe was either wood, stone, iron, or diamond. The figures show the results for systems with $n_a = 2$. Showing the results for a single agent system, with $n_a = 1$, is redundant as the difference between the systems was trivial. All agents in the system started with the same inventory. The points show the average completion time for the different methods. The blue dots show the average time for the traditional backward chained algorithm as shown in Algorithm 1. The red dots show the average time for collaborative backward chaining without backup as presented in Algorithm 2. The green dots show the average time for collaborative backward chaining without backup as presented in Algorithm 3. The figures also include error bars denoting the standard deviation.

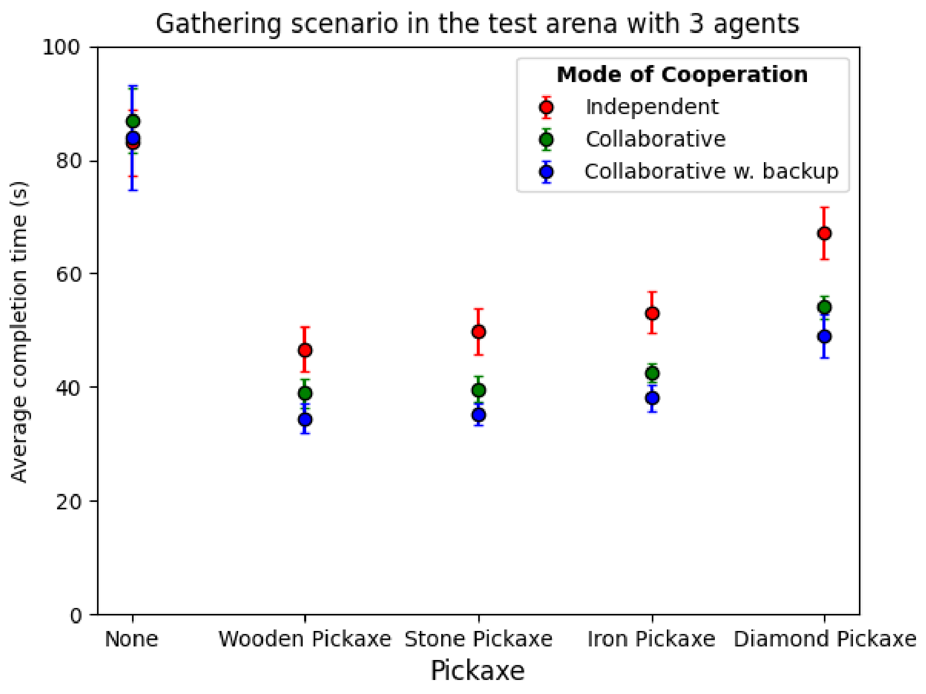


Figure 7.14: Average completion time for the stone pickaxe scenario in the authentic world for three agents starting with different inventories. Each figure depicts the average completion time from 15 trials of each parameter combination; system, starting inventory, and the number of agents. The agents could either have no items or start with a single pickaxe. The material of the pickaxe was either wood, stone, iron, or diamond. The figures show the results for systems with $n_a = 3$. Showing the results for a single agent system, with $n_a = 1$, is redundant as the difference between the systems was trivial. All agents in the system started with the same inventory. The points show the average completion time for the different methods. The blue dots show the average time for the traditional backward chained algorithm as shown in Algorithm 1. The red dots show the average time for collaborative backward chaining without backup as presented in Algorithm 2. The green dots show the average time for collaborative backward chaining without backup as presented in Algorithm 3. The figures also include error bars denoting the standard deviation.

n_t	15
n_a	$[1, 2, 3]$
$\mathcal{S}$	$[\mathcal{S}_I, \mathcal{S}_C, \mathcal{S}_{CC}]$
$\mathbf{P}_a$	$[[131, 72, 17], [132, 72, 4], [140, 72, 24]]$
$\mathcal{I}$	$\mathcal{I}_P$
n_p	5
s	4000020

Table 7.8: Hyperparameters used by the stone pickaxe scenario with variable starting inventory in the test arena. Here, n_t is the number of trials, n_a, the number of agents, $\mathcal{S}$ the system, $\mathbf{P}_a$, the agents' initial positions, $\mathcal{I}$, the starting inventory, n_p, the number of pickaxes to gather materials for, s, the generation seed, and G, the generator configuration string. Here $\mathcal{I}_P =$[None, A wooden pickaxe, A stone pickaxe, An iron pickaxe, A diamond pickaxe] is the set of pickaxes.

Figure 7.6 shows that when agents place fences in the flat world, the mission's success rate benefits from agents having dynamic conditions that consider other agents. We can also see that several agents working together increases the success rate.

Figure 7.7 shows that the average completion time of the experiment is mostly unchanged. We can see that the completion time benefits from observant agents when there are two agents. The surviving agents work together to complete the mission faster. But when there are three agents, the completion time increases from the observant agents. Probably because when the zombie attacks an agent, both other agents run to help, even if it would be enough if only one of them came to the rescue. This could have been circumvented by using the n-to-1 task distribution mentioned in Section 5.2.3.

The value of helpful agents is tightly coupled to the opportunity cost for an agent to help another agent rather than attempting to achieve all goals alone. In the scenario, several agents place fence posts at nine target points. A patrolling zombie equipped with a stone sword guards the target points. The zombie has a high chance of defeating a lone agent but a low chance of defeating two or more attacking it simultaneously. As the agents run around placing fence posts, the zombie eventually ambushes one of them. If the ambushed agent is alone, the zombie will likely win and continue to patrol the target points. Then, if the zombie defeats the agents one by one, the MAS fails. Thus, when an agent gets ambushed, the reactions of other agents are critical. If a nearby agent helps the ambushed agent, they can defeat the zombie together and get rid of it for the rest of the experiment trial. In the scenario, the opportunity cost of helping another agent is small since completing the goals is difficult without defeating the zombie first.

On the other hand, in a scenario where the agents can achieve the goals more independently, the value of helpful agents is lower. For example, the agents are gathering materials, but only one of the agents is at risk of getting confronted by the zombie. If it is easy for the other agent to help, defeating the zombie decreases the completion time of the MAS as both agents get more time to work towards the goal. But, if it is difficult to help, it might be more efficient for the agent to leave its companion to its demise and continue towards completing the goal.

7.2.2 Evaluation of collaborative backward chaining with backup

The results of the two scenarios confirmed some presumptions of backward chaining with 1-to-1 task distribution. First, 1-to-1 task distribution benefits from a significant goal-switching cost (defined in Section 5.3.2) and works better for unshareable goals than shareable ones. Second, collaboration with backup (Algorithm 3) provides the best outcome in both scenarios. Furthermore, in the fence post scenario, the larger Δ in the FWG world increases the goal-switching cost, giving the proposed technique a more pronounced advantage over the reference case when achieving unsharable goals.

Meanwhile, the primary learning from the stone pickaxe scenario is the importance of backup to avoid idling agents. Executing the stone pickaxe scenarios with a variable inventory showed the benefits of 1-to-1 task distribution in scenarios with high goal-switching costs. Both scenarios show how collaborative backward chaining fulfills the goals stated in Section 1.4.

Overview

Figure 7.1 and 7.2 shows a significant completion time reduction when using collaborative backward chaining (Algorithm 2 or Algorithm 3) instead of traditional backward chaining (Algorithm 1), which indicates the benefit of 1-to-1 task distribution in the fence post placement scenario. The significant efficiency improvements from Algorithm 2 or Algorithm 3 has three explanations:

- The tasks are unshareable. Several agents working together on the same unsharable goal means redundant extra work with minimal benefits. Since it only matters whether the agents place a single fence post at the intended position, several agents attempt simultaneously to reach the target position of the first unfulfilled goal in the goal set. The agents unnecessarily heading toward the same position can be seen in Figure 7.10, which depicts the paths of the agents. The entire system of agents will never achieve the goal faster than the agent closest to the target position. Contrarily, the agents may block each other, delay the completion time, or even make the system fail in achieving the goals. These results are supported by Theorem B.1.
- The agents have a high cost for switching between tasks. Running towards one target position typically moves the agents further away from

the other positions. Also, as seen by comparing 7.2 and 7.1, the impact from switching tasks is particularly significant in the world generated by the FWG, where the distance between the fence posts is larger than in the world created by the DWG.

- There are many goals that all have a similar completion time. There are 9 goals, giving each agent 3 goals even with the maximum $n_a = 3$. The considerable number of goals and the absence of significant time disparities reduce the portion of idling.

The impact of Δ, the distance between the fence posts, on the goal-switching costs is further illustrated by Figure 7.4, which shows the completion time of systems with three agents for placing fence posts at different values of Δ. Figure 7.5 shows how the completion time of $\mathcal{S}_C$ and $\mathcal{S}_{CC}$ increases relative to $\mathcal{S}_I$ as the agents put the fence posts further from each other.

On the other hand, 1-to-1 task distribution negatively influences the completion time in Figure 7.8, which reveals the weaknesses of collaborative backward chaining without backup (Algorithm 2). Contrarily to the fence post scenario, the stone pickaxe scenario has shareable goals and only two goals with a significant completion time difference. As presented theoretically in Theorem B.9, systems of shareable goals naturally favor traditional backward chaining (Algorithm 1) over the collaborative extension Algorithm 2. Additionally, the goal-switching cost is low in the authentic world, leaving little room for collaboration. However, Figure 7.9 shows that 1-to-1 task distribution can benefit agents collaborating in completing shareable goals in the manually created test arena because of the higher goal-switching costs. The impact of the goal-switching costs is investigated further in Section 5.3.2.

Figure 7.8 and Figure 7.9 also show the significance of adding backup to remove idling. Mainly, illustrating that the completion time of the simple collaborative extension (Algorithm 2) is unaffected by adding a third agent indicates that the added agent idles instead of working on a task.

Looking at all figures reveals the differences between the performance in the two scenarios and several similarities. Collaboration with backup (Algorithm 3) performs equally well as or better than traditional backward chaining (Algorithm 1)and collaboration without backup (Algorithm 2) in all experimental setups. Thus, collaboration with backup is efficient and accurate in both the best and worst circumstances for 1-to-1 task distribution.

Conclusively, all results point towards collaborative backward chaining with backup (Algorithm 3) achieving all three project goals defined in Section

1.4.

- Algorithm 3 (collaborative backward chaining with backup) outperforms Algorithm 1 (traditional backward chaining) in scenarios that strongly benefit from 1-to-1 task distribution. And when 1-to-1 task distribution is suboptimal, Algorithm 3 (collaborative backward chaining with backup) performs at least equally well as Algorithm 1 (traditional backward chaining).
- For a single agent, there is no impact on the accuracy or efficiency of the MAS in using Algorithm 3 (collaborative backward chaining with backup) rather than Algorithm 1 (traditional backward chaining).
- A larger number of agents is not detrimental for he efficiency of Algorithm 3 (collaborative backward chaining with backup). Contrary, the efficiency improves with more agents [1].

Variable start inventory in the test arena

Varying the starting inventory altered the goal-switching cost, which directly impacted the efficiency of using collaboration with backup over traditional backward chaining.

Figure 7.11 and Figure 7.12 depict the average completion time for agents with different starting inventories in the test arena. The results demonstrate that 1-to-1 task distribution can be beneficial in scenarios with shareable goals when there are considerable goal-switching costs.

Figure 7.11 and Figure 7.12 show that when the agents start empty-handed, the naive collaborative method (Algorithm 2) performs weaker than traditional backward chaining (Algorithm 1). Nonetheless, collaboration with backup (Algorithm 3) performs equally well as traditional backward chaining (Algorithm 1). Figure 7.11 and Figure 7.12 also show how the naive collaborative method ((Algorithm 2) outperforms traditional backward chaining (Algorithm 1). Separating the materials in the test arena increases the goal-switching costs. However, if the system has a dedicated stone gatherer and a wood gatherer, the stone gatherer's BT enforces it to gather wood to craft the required wooden pickaxe before it collects stone. The stone gatherer's need to gather wood mitigates the goal-switching cost from the separation of the wood and stone sources in the test arena. Nonetheless, if both agents start

[1] Due to performance constraints (see Section 1.6), there were at most five agents. That the solution is scaleable for even larger amounts of agents is assumed inductively (while considering the linear agent complexity).

with a pickaxe, independent of the pickaxe quality, the stone gatherer can go directly to the stone source.

Figure 7.10 shows the paths of the agents starting with a wooden pickaxe in the test arena when gathering stone and wood. Although the wood gatherer reaches the stone chunk, it is first after it has finished gathering wood, thus contributing very little to the stone gathering. The stone gatherer, on the other hand, goes directly to the stone source and stays there until it has enough stone.

The increased goal-switching cost allows collaborative backward chaining without backup (Algorithm 2) to outperform traditional backward chaining (Algorithm 2). Additionally, collaboration with backup (Algorithm 3) outperforms both methods since it utilizes the full benefit of task distribution while avoiding idle agents.

On the other hand, Figure 7.13 and Figure 7.14 show that giving the agents a pickaxe does not improve the results in the authentic DWG world. Instead, the traditional backward chaining method (Algorithm 1) outperforms the naive collaborative method (Algorithm 2). Furthermore, collaboration with backup (Algorithm 3) performs equally well as traditional backward chaining (Algorithm 2) when considering statistical significance.

Chapter 8

Conclusions and Future work

The project met all goals defined in Section 1.4. Our proposed solution, Algorithm 3, was a good solution for improved collaboration among agents in a MAS. The method was also investigated in detail, showing the strategy's pros and cons. The thesis also revealed some edge cases where the approach works less well. Finally, there is a clear understanding of the use cases of the strategy, which aids in expanding it for future work.

This chapter presents the conclusions in more detail in four sections. First, Section 8.1 reveals general findings from the research. Second, Section 8.2 exposes unexpected delimitations. Third, Section 8.3 presents opportunities to continue the research. Finally, Section 8.4 reflects on the learnings of the project and its value in the context of scientific research.

8.1 Conclusions

The fundamental research question of this project was to find solutions for multi-agent coordination in complex adversarial environments using backward chained BTs (see Section 1.2). The results from Chapter 7 show that, while backward chained BTs perform well in complex adversarial environments, there are multiple methods to improve their efficiency and success rate in MASs. This project investigated two; reactively cooperative agents assessing the threat risks of other agents (Chapter 4) and 1-to-1 task distribution (Chapter 5).

Our intention, as stated in Section 1.4, was to find a method that was transferable enough that it could be easily used with pre-existing PPA-BTs or PPA sets. In that regard, 1-to-1 task distribution is a better choice than reactive cooperation. Although reactive cooperation increases the success rate of the MAS (see Figure 7.6), it requires manual editing of the behavior nodes to enable observant agents. Additionally, reactive cooperation transferability (in this case, the ability to improve cooperation of agents already using non-collaborative backward chained BTs) is negatively affected by requiring a more extensive observation state than non-collaborative solutions. 1-to-1 task distribution, on the other hand, is possible to apply on any pre-existing PPA-BTs or PPA sets to increase the coordination ability of the agents.

The proposed approach, collaboration with backup (Algorithm 3), was more efficient and accurate than independent agents using traditional backward chaining (Algorithm 1) in both scenarios. The fence post scenario got good results with the unrefined collaborative backward chaining (Algorithm 2) from Section 5.2. In contrast, the stone pickaxe scenario relied on the sophisticated method using backup (Algorithm 3) presented in Section 5.4.

Collaboration through 1-to-1 task distribution works best when there is no benefit for several agents to simultaneously work on the same task. For example, when placing a fence post, obtaining a single item, or hunting down an NPC. An example of a shareable task is

$$G(I, n) = \textbf{Agents have n I}, \tag{8.1}$$

used in the stone pickaxe scenario. This objective enables several agents to work towards it simultaneously to decrease the completion time. However, these shareable tasks might not be very common. Most conditions are either too dependent on the agents' individual states, or there is no improvement from several agents working together to achieve them. An example of a goal that only depends on a single agent's state while being independent of other agents' state in the MAS is

$$G_i = \textbf{Has a stone pickaxe in inventory}. \tag{8.2}$$

And an example of a goal that is not faster to achieve with more agents working towards it simultaneously is

$$G_f = \textbf{Is fence post at position [0, 65, 0]}. \tag{8.3}$$

The refined proposal, collaboration with backup, as presented in Algorithm 3, removes idling agents and thus handles any potential issues with the shareable objective edge cases.

Another possible workaround is to replace each $G(I, n)$ with $n\ \delta G_\delta(I, \nu)$

where

$$G_\delta(I, \nu) = \textbf{Agents have } \nu \ \mathbf{I}, \nu \in [1, n]. \tag{8.4}$$

This replacement works well but relies on more advanced methods for distributing the goals than distributing them in order. Distributing these tiny goals in order disables the collaborative approach, as both agents simultaneously would gather first wood and then stone.

Backward chained BTs performs well in the context of sudden threats such as hostile entities. Although adopting the behaviors to handle a varying amount of agents dynamically is beneficial, it is not necessary. The reactive nature of BTs enables agents to react rapidly to sudden hazards and then continue with their primary objectives.

A significant challenge was to design the method so it was generic, and while it partly succeeded, it is still an open problem to increase the generality of the strategy. One limitation of the method is the need to specify which preconditions to incorporate into the 1-to-1 task distribution scheme. However, most importantly, extending genericity requires more experimentation, with task collaboration deeper in the tree. Both scenarios were limited to distributing the goals in $\vec{G}$.

Task distribution may have performed even better with a belief system or another procedure to distribute the tasks closer to the optimal task distribution. In the proposed solution, the collaborative agents require an active communication channel, the blackboard, to distribute the tasks. The requirement of inter-agent communication could limit the agents' ability to cooperate with a human player.

8.2 Limitations

Some limitations were discovered during the project. The largest was that the Minecraft game loop runs independently of the Project Malmo control (BT ticking) loop. So instead of waiting for the next command, the world updates continuously, and agents automatically proceed with continuous actions.

8.2.1 Asynchronicity between game and controller

If the control loop runs faster than the game loop without waiting for new observation states, the control loop can unintentionally evaluate the same observation state more than once. Thankfully a too-fast control loop can be avoided entirely by using Project Malmo to wait for new observations. Only the opposite case, where the controller falls behind the game loop, posed a risk. Specifically, a lagging control loop is harder to avoid and more significantly impacts BTs.

Ticking a BT slower than the game loop can negatively and seemingly positively impact its accuracy. The seemingly positive consequence is that if the BT is purposely slower than the game, it can send a series of commands while ticking a single action. It can also wait a preset time by sleeping the thread on which the BT runs. Waiting between commands is beneficial if the commands take a long time to execute.

Purposely running the BT slower than the game loop may seem like a good idea, but a deliberately slow BT is a faulty design pattern. The slow actions stop the rest of the tree from being ticked, which ultimately diminishes reactivity. Blocking the ticking makes the agent miss important observation data.

Slow ticks can emerge unintentionally due to performance issues. Unintentional lagging can have an even more detrimental effect than intentional tardiness. When controlling a Minecraft agent with continuous actions through Project Malmo, the ongoing actions need to be able to start and stop promptly. Otherwise, the agent can keep moving or turning too far. If the performance issue occurs frequently, the agent can not progress. To avoid these side effects, the BT's ticking speed required an upper limit equal to the game's ticking rate. Staying below the limit required careful observations and evaluations of the performance of the conditions and actions. When the backward chained trees are gigantic, it traverses many behaviors during each cycle.

8.2.2 Behavior performance optimization

Due to the shape of the PPA-BT, each cycle, the BT ticks several more conditions than actions. On top of that, most actions spend several cycles in the **Running** state. Because of these two reasons, more conditions are ticked than actions, which puts a stricter performance requirement on conditions than on actions. Therefore, several conditions had temporary caches, invalidated during each controller cycle, to reduce condition execution time.

Project Malmo allowed reducing the game ticking interval to speed up the evaluation time of the experiments. The performance of the behaviors was optimized enough to double the ticking speed for a single agent. Nevertheless, reducing game ticking time was impossible for systems with multiple agents. Therefore, all experiments ran in real game time. The single-agent experiments did as well to ensure posterity across experiments.

The BT did not cause the performance issues. The reason was that Minecraft's RAM usage scaled poorly with several agents. The performance issues could have been circumvented by spreading the agents across several machines. However, distributing the implementation across several devices would require extensive changes. In the experimental setup, the agents communicate with Python's multiprocessing library, which enables inter-process communication limited to a single machine. Therefore, due to performance issues, the automatic evaluations were limited to groups of three.

The asynchronicity between the control loop and the game loop is a design flaw of Project Malmo. The interface can limit game execution time to a preset time interval but cannot give the control system complete control over game execution cycles.

8.3 Future work

The findings serve as a basis for plenty of future work. It would be interesting to compare the results with other MAS experiments that have used Minecraft as the experimental environment. There are also multiple ways to extend the work of the thesis with RL. RL could be used for substituting the blackboard communication with an observation-based belief system or to optimize the distribution of the tasks. Additionally, n-to-1 task distribution could be solidified with an experiment.

Comparing the results with other MAS methodologies in Minecraft would be beneficial, but finding other methods to evaluate similar scenarios was challenging. For example, the RL experiments which also focused on MAS in Minecraft, had too different execution rules and evaluation metrics.

Additionally, using a decentralized MAS instead of a distributed MAS would have been interesting. A decentralized MAS is possible by replacing the blackboard with a belief system such as fuzzy signatures [37]. However, Fuzzy signatures would require RL, which would have drastically changed the focus and scope of the thesis. Thus, using a blackboard worked well for displaying the advantages of introducing task distribution to backward chaining of BTs when constructing agents that cooperate in highly dynamic environments.

RL could also help other aspects of the method, such as determining which conditions that are easiest to distribute and which conditions that benefit from reactive cooperation. Any approach to dynamically discover the distributable preconditions would have reduced the overhead for the user of the algorithm. Additionally, in the proposed solution, the agents reserve each condition linearly. Job shop scheduling or other methods to decide the order of the distributable conditions could have improved the result. It would have also made the methodology more valuable.

The proposed solution uses 1-to-1 task distribution, but as mentioned in Section 5.2.3, it would be trivial to use n-to-1 task distribution instead. However, because of performance limitations (see Section 8.2), the experiments presented in Chapter 7 have two or three agents. A larger number of agents is required for there to be a substantial difference between a task that is distributed to n agents and a task that has distributability $D = 0$.

8.4 Reflections

Although the collaborative methods were evaluated in Minecraft, the techniques were general and thus applicable to game development, robotics, and other fields that utilize AI and benefit from collaborative agents. Since collaborative robots are more efficient than independent robots, fewer robots are required for automatizing advanced multi-robot tasks. Manufacturing fewer robots has both environmental and economic advantages. Additionally, robots that can collaborate and interact with both other robots and humans will have a substantial social impact. Finally, while collaboration can make society more sustainable by increasing the efficiency of the production and distribution of goods, collaborative robots can also be used to harm people.

The work shows new ways of using BTs together in a MAS. All assumptions were to ensure generality for the results to be applied elsewhere within game development, robotics, and other fields that utilize AI. The method is especially straightforward for systems that already use backward chained BTs but want to add more agents to the setup.

Systems of collaborative robots contain fewer robots than systems of independent robots, thus lowering robot manufacturing needs. Fewer manufactured robots have both a positive environmental effect and a positive economic effect. Environmentally, manufacturing fewer robots requires fewer resources. Economically, improving efficiency through collaboration and other software improvements, instead of just constructing more robots, lead to long-term cost savings.

The advent of more collaborative robots in society, especially ones apt at human collaboration, will have an enormous social impact. For example, imagine that you work at a construction site. A group of collaborative robots works together at the site to construct structures. You interact with them regularly through a feedback system as part of your job. At first, this artificial interaction will feel new and strange. But, as you grow used to it, your relationship with the collaborative robots may mimic that of a farmer and a work animal. You will start seeing the interactions between the robots as dogs working together to pull a snow sled. In this example, the robots are relatively simple, with a limited ability to collaborate with both you and each other. As the robots become more advanced in their interaction with you, your relationship with the robots may take the form of coworkers. However, as shown in the report, inter-agent collaboration is easier than agent-human collaboration. This means that, while your relation to the robot has grown to be able to discuss work tasks in detail and give each other clear feedback, the inter-robot collaboration ability is at another level.

How collaborative robots are used is essential from an ethical perspective. Using collaborative robots in manufacturing or logistics can help create a more sustainable society. However, using collaborative robots in, for example, warfare raises a moral dilemma. A lot of research in creating advanced algorithms for robotics risks is being used in optimizing the automation of harming people. It is, therefore, critical to keep in mind when developing new algorithms to condemn using them to violate human rights.

References

[1] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8839–8845. [Pages 1, 23, and 24.]

[2] Microsoft, "The malmo collaborative ai challenge," Mar 2020. [Online]. Available: https://www.microsoft.com/en-us/research/academic-program/collaborative-ai-challenge/ [Pages 3 and 25.]

[3] J. Ota, "Multi-agent robot systems as distributed autonomous systems," *Advanced Engineering Informatics*, vol. 20, no. 1, pp. 59–70, 2006. doi: https://doi.org/10.1016/j.aei.2005.06.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474034605000509 [Page 4.]

[4] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018. [Pages 4, 5, and 22.]

[5] O. Biggar, M. Zamani, and I. Shames, "On modularity in reactive control architectures, with an application to formal verification," *ACM Transactions on Cyber-Physical Systems (TCPS)*, vol. 6, no. 2, pp. 1–36, 2022. [Page 4.]

[6] R. A. Agis, S. Gottifredi, and A. J. García, "An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games," *Expert Systems with Applications*, vol. 155, p. 113457, 2020. [Pages 6, 24, and 35.]

[7] S. Jones, M. Studley, S. Hauert, and A. Winfield, "Evolving behaviour trees for swarm robotics," in *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 487–501. [Pages 6, 22, 24, and 42.]

[8] B. Games, "Halo 2," 2004. [Page 9.]

[9] K. Games, “Bioshock,” 2007. [Page 9.]

[10] “Behavior trees.” [Online]. Available: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/ [Page 10.]

[11] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, “Towards a unified behavior trees framework for robot control,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014. doi: 10.1109/ICRA.2014.6907656 pp. 5420–5427. [Page 10.]

[12] J. Ferber and G. Weiss, *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading, 1999, vol. 1. [Page 15.]

[13] C. Peng, *Optimization Based Control for Multi-agent System with Interaction*. University of California, Berkeley, 2019. [Page 16.]

[14] E. Bahceci, O. Soysal, and E. Sahin, “A review: Pattern formation and adaptation in multi-robot systems,” *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-43*, 2003. [Page 16.]

[15] H. Ma, J. Yang, L. Cohen, T. S. Kumar, and S. Koenig, “Feasibility study: Moving non-homogeneous teams in congested video game environments,” in *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017. [Page 16.]

[16] T. M. Cheng and A. V. Savkin, “Decentralized control of multi-agent systems for swarming with a given geometric pattern,” *Computers & Mathematics with Applications*, vol. 61, no. 4, pp. 731–744, 2011. doi: https://doi.org/10.1016/j.camwa.2010.11.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0898122110008953 [Page 16.]

[17] A. Campbell and A. S. Wu, “Multi-agent role allocation: issues, approaches, and multiple perspectives,” *Autonomous agents and multi-agent systems*, vol. 22, no. 2, pp. 317–355, 2011. [Pages 16, 17, and 40.]

[18] ——, “On the significance of synchroneity in emergent systems.” in *AAMAS (1)*. Citeseer, 2009, pp. 449–456. [Page 17.]

[19] A. Gage and R. R. Murphy, “Affective recruitment of distributed heterogeneous agents,” in *AAAI*, 2004, pp. 14–19. [Pages 17 and 35.]

[20] B. P. Gerkey and M. J. Mataric, “A framework for studying multi-robot task allocation,” 2003. [Pages 17 and 39.]

[21] A. Farinelli, L. Iocchi, D. Nardi, and F. Patrizi, “Task assignment with dynamic token generation,” in *Monitoring, Security, and Rescue Techniques in Multiagent Systems*. Springer, 2005, pp. 467–477. [Page 17.]

[22] M. J. Mataric, “Minimizing complexity in controlling a mobile robot population,” in *Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 1992, pp. 830–831. [Page 17.]

[23] T. H. Labella, M. Dorigo, and J.-L. Deneubourg, “Division of labor in a group of robots inspired by ants’ foraging behavior,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 1, pp. 4–25, 2006. [Page 17.]

[24] K. Lerman, C. Jones, A. Galstyan, and M. J. Matarić, “Analysis of dynamic task allocation in multi-robot systems,” *The International Journal of Robotics Research*, vol. 25, no. 3, pp. 225–241, 2006. [Page 17.]

[25] K. Yanai and H. Iba, “Multi-agent robot learning by means of genetic programming: Solving an escape problem,” in *International Conference on Evolvable Systems*. Springer, 2001, pp. 192–203. [Page 17.]

[26] G. Baldassarre, S. Nolfi, and D. Parisi, “Evolving mobile robots able to display collective behaviors,” *Artificial life*, vol. 9, no. 3, pp. 255–267, 2003. [Page 17.]

[27] Mojang, “Minecraft,” 2011. [Page 17.]

[28] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, “The malmo platform for artificial intelligence experimentation,” in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI’16. AAAI Press, 2016. ISBN 9781577357704 [Pages 17 and 19.]

[29] A. Grow, M. Dickinson, J. Pagnutti, N. Wardrip-Fruin, and M. Mateas, "Crafting in games," *Digital Humanities Quarterly*, vol. 11, no. 4, 2017. [Page 18.]

[30] "Pickaxe." [Online]. Available: https://minecraft.fandom.com/wiki/Pickaxe [Page 18.]

[31] C.-U. Lim, R. Baumgarten, and S. Colton, "Evolving behaviour trees for the commercial game defcon," in *European conference on the applications of evolutionary computation*. Springer, 2010, pp. 100–110. [Page 22.]

[32] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving behaviour trees for the mario ai competition using grammatical evolution," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2011, pp. 123–132. [Page 22.]

[33] P. McClarron, R. Ollington, and I. Lewis, "Effect of constraints on evolving behavior trees for game ai," in *International Conference on Computer Games, Multimedia & Allied Technology (CGAT). Proceedings*. Global Science and Technology Forum, 2016, p. 1. [Page 22.]

[34] J. Á. Segura-Muros and J. Fernández-Olivares, "Integration of an automated hierarchical task planner in ros using behaviour trees," in *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. IEEE, 2017, pp. 20–25. [Page 23.]

[35] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Combining planning and learning of behavior trees for robotic assembly," *arXiv preprint arXiv:2103.09036*, 2021. [Page 23.]

[36] Y.-M. De Hauwere, P. Vrancx, and A. Nowe, *Multi-Agent Systems and Large State Spaces*, 07 2010, vol. 289, pp. 181–205. [Page 24.]

[37] A. Ballagi, L. Koczy, and T. Gedeon, "Robot cooperation without explicit communication by fuzzy signatures and decision trees." 01 2009, pp. 1468–1473. [Pages 25 and 103.]

[38] Á. Ballagi and L. T. Kóczy, "Robot cooperation by fuzzy signature sets rule base," in *2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2010, pp. 37–42. [Page 25.]

[39] Y. Xiong, H. Chen, M. Zhao, and B. An, “Hogrider: Champion agent of microsoft malmo collaborative ai challenge,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018. [Page 25.]

[40] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” *Autonomous Robots*, vol. 8, pp. 345–383, 2000. [Pages 35 and 47.]

[41] C. Sun, P. Karlsson, J. Wu, J. B. Tenenbaum, and K. Murphy, “Stochastic prediction of multi-agent interactions from partial observations,” *arXiv preprint arXiv:1902.09641*, 2019. [Page 36.]

[42] L. B. Johnson, H.-L. Choi, and J. P. How, “The role of information assumptions in decentralized task allocation. a tutorial,” *IEEE Control Systems Magazine*, vol. 36, no. 4, pp. 45–58, 2016. doi: 10.1109 MCS.2016.2558419 [Page 40.]

[43] “Minecraft superflat preset generator.” [Online]. Available: https://www.minecraft101.net/superflat/ [Pages 60 and 74.]

Appendix A

Behavior trees

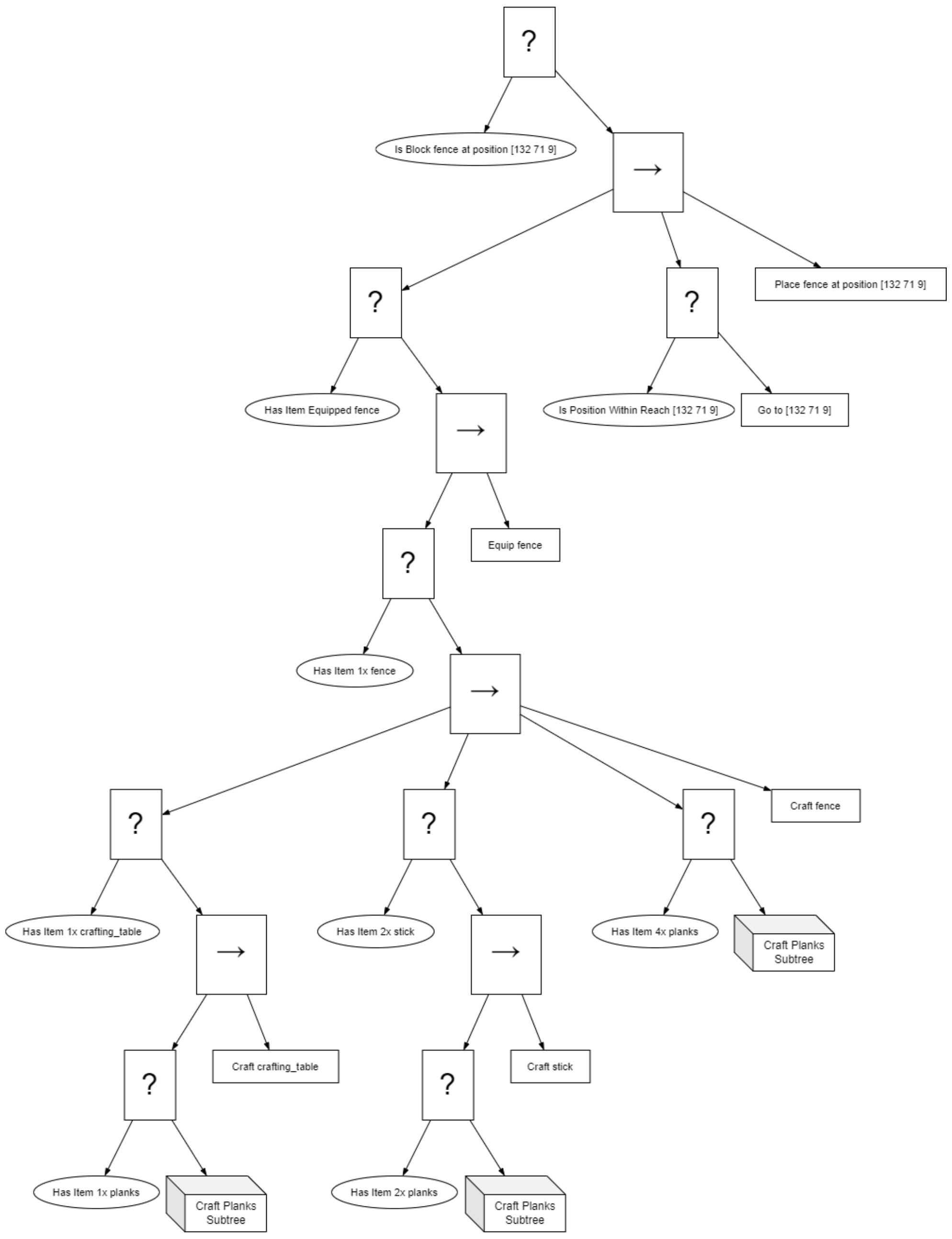


Figure A.1: The tree created from backward chaining from the postcondition **Is Block fence at position** $[\mathbf{132}, \mathbf{71}, \mathbf{9}]$. The ”Craft Planks” subtree can be seen in Figure A.3.

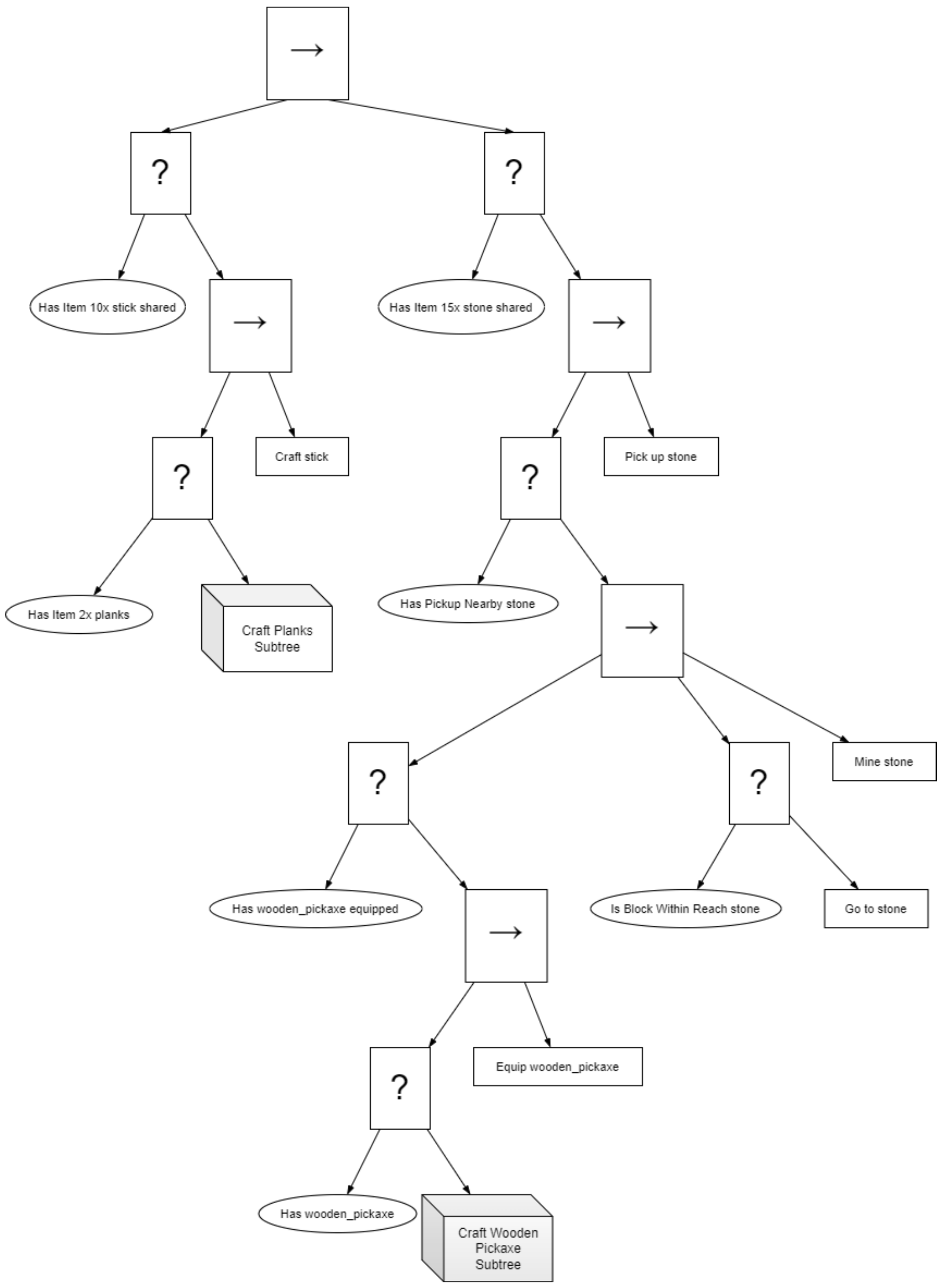


Figure A.2: The tree created from backward chaining from the goals [**Has Item 10x Sticks Shared**, **Has Item 15x Stone Shared**]. The craft planks subtree can be seen in Figure A.3, and The craft wooden pickaxe subtree can be seen A.4.

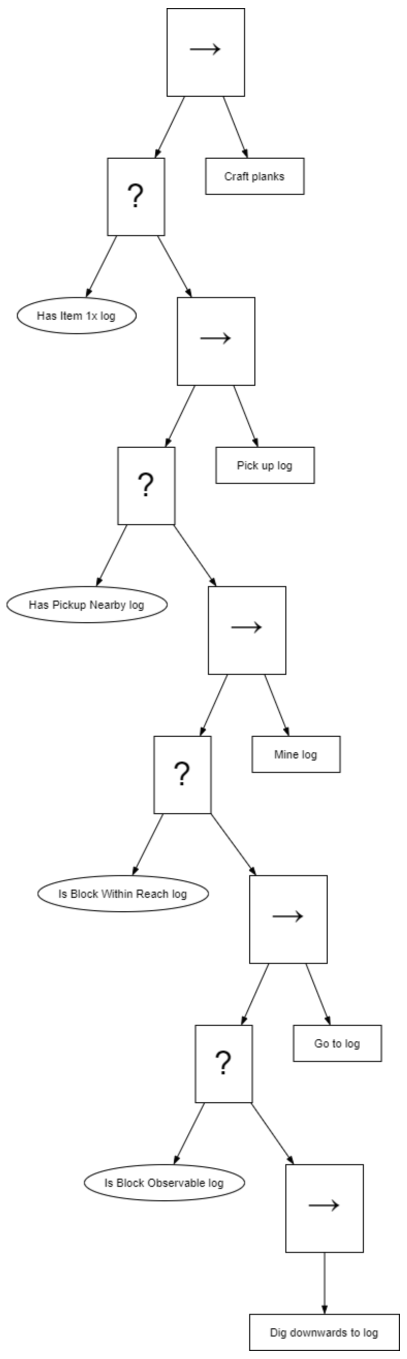


Figure A.3: The tree created from backward chaining for crafting planks. This is used in Figure A.1 and Figure A.2.

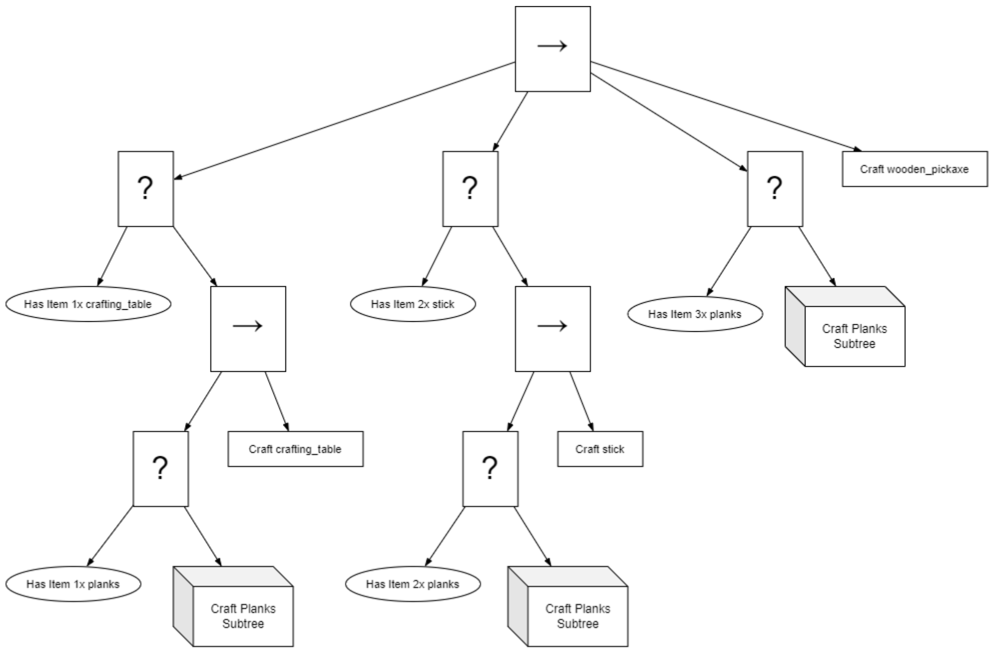


Figure A.4: The tree created from backward chaining for crafting a wooden pickaxe. This is used in Figure A.2.

TRITA-EECS-EX-2023:547

For DIVA

{
”Author1”: { ”Last name”: ”Salér”,
”First name”: ”Justin”,
”Local User Id”: ”justinr”,
”E-mail”: ”justinr@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
”Cycle”: ”2”,
”Course code”: ”DA233X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ”Master’s Programme, Machine Learning, 120 credits”
,”programcode”: ”TMAIM”
,”Degree”: ”Master’s Programme, Machine Learning, 120 credits”
,”subjectArea”: ”Computer Science and Engineering”
},
”Title”: {
”Main title”: ”Using Backward Chained Behavior Trees to Control Cooperative Minecraft Agents”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”Användning av bakåtkedjade beteendeträd för att kontrollera samarbetande agenter i Minecraft”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Ögren”,
”First name”: ”Petter”,
”Local User Id”: ”tbd”,
”E-mail”: ”petter@kth.se”,
”organisation”: {”L1”: ””,
”L2”: ”Robotics, Perception and Learning” }
},
”Examiner1”: { ”Last name”: ”Smith”,
”First name”: ”Christian”,
”Local User Id”: ”tbd”,
”E-mail”: ”ccs@kth.se”,
”organisation”: {”L1”: ””,
”L2”: ”Robotics, Perception and Learning” }
},
”National Subject Categories”: ”10201, 10207”,
”Other information”: {”Year”: ”2023”, ”Number of pages”: ”1,113”},
”Series”: { ”Title of series”: ”TRITA-EECS-EX” , ”No. in series”: ”2022:00” },
”Opponents”: { ”Name”: ”Ramtin Erfani Torbaghani”},
”Presentation”: { ”Date”: ”2023-06-08T13:00:00+01:00”
,”Language”:”eng”
,”Room”: ”Room 304”
,Äddress”: Teknikringen 14”
,”City”: ”Stockholm” },
”Number of lang instances”: ”2”,
”Abstract[eng]”: €€€€

This report presents a strategy to control multiple collaborative intelligent agents acting in a complex, versatile environment. The proposed method utilizes back-chained behavior trees and 1-to-1 task distribution. The agents claim a task, which prevents other agents in the system to start working on the same task. Backward chaining is an algorithm for generating reactive agents from a set of goals. The method was evaluated in Minecraft with Microsoft’s Project Malmo API. Two different scenarios were considered. In the first one, a group of agents collaborated to build a structure. In the second one, a group of agents collaborated while gathering material. We propose and evaluate three algorithms with different levels of cooperation (Algorithm \ref{alg:bc}, Algorithm \ref{alg:bcmod}, and Algorithm \ref{alg:bcfinal}). The evaluation shows that backward chained \glspl{BT} works well for multi-agent coordination in complex versatile environments and that adding 1-to-1 task distribution increases the efficiency of the agents further when completing the experiment tasks.

€€€€,
”Keywords[eng]”: €€€€
Behavior-Based Systems, Multi-Agent Systems, Behavior Trees, Minecraft €€€€,
”Abstract[swe]”: €€€€

Rapporten presenterar en metod för styrning av en grupp kollaborativa intelligenta agenter agerande i en komplex dynamisk miljö. Den förslagna metoden använder sig av bakåtkedjade beteendeträd och 1-mot-1 uppgiftsdistribution, där en agent reserverar en uppgift vilket hindrar andra agenter att börja arbeta på samma uppgift. Bakåtkedjning är en metod som möjliggör generering av flexibla agenter utifrån en lista av mål och krav. Metoden utvärderades i två olika scenarion i tv-spelet Minecraft. Agenterna samarbetar i det första scenariot med att bygga en struktur och i det andra scenariot med att samla material. Vi föreslår och utvärderar tre algoritmer med olika nivåer av agentsamarbete och komplexitet (Algoritm \ref{alg:bc}, Algoritm \ref{alg:bcmod}, och Algorithm \ref{alg:bcfinal}). Utvärderingerarna indikerar att bakåtkedjade beteendeträd fungerar bra för multiagentkoordination i komplexa dynamiska miljöer och att 1-mot-1 uppgiftsdistribution ökar agenternas förmåga att genomföra experimentuppgifterna ytterligare.

€€€€,
”Keywords[swe]”: €€€€
Beteendebaserade system, multiagentssystem, beteendeträd, Minecraft €€€€,
}

Appendix B

Additional time execution estimations

The execution times for the two different approaches can be estimated to show the advantages and flaws of collaborative backward chaining (Algorithm 2) in comparison to traditional backward chaining (Algorithm 1). For a set of unshareable goals (Definition 5.3), in the absence of any significant goal-switching cost, collaborative backward chaining (Algorithm 2) performs equally well as or better than traditional backward chaining (Algorithm 1). Contrarily, it can be shown that for a set of shareable goals (Definition 5.1), in the absence of any significant goal-switching cost, collaborative backward chaining (Algorithm 2) performs equally well as or worse than traditional backward chaining (Algorithm 1). Finally, for a set of shareable goals (Definition 5.1), with significant goal-switching costs, collaborative backward chaining (Algorithm 2) can perform better than traditional backward chaining (Algorithm 1).

B.1 Unshareable goals

We can prove that for unshareable goals, as in Definition 5.3, when not considering any goal-switching costs, collaborative backward chaining (Algorithm 2) performs equally well as or better than traditional backward chaining (Algorithm 1) We will show that Algorithm 2 performs equally well as or better than Algorithm 1.

Our proofs are performed through several lemmas and theorems. First, we prove that, given a set of unshareable goals with a constant completion time, the completion time for agents using Algorithm 2 is shorter than for agents using Algorithm 1. Second, we prove that the same holds for sets of unshareable goals with variable completion times. The proof concerning unshareable goals with variable completion times is also separated into two cases; when there are more goals than agents and when there are more agents than goals.

B.1.1 Constant goal completion time

We will first prove that Algorithm 2 gives better completion times than Algorithm 1 for sets of unshareable goals with the same completion time but also show it without that assumption.

Theorem B.1. *Given a set of unshareable goals with the same completion time, the completion time for agents using Algorithm 2 is shorter than for agents using Algorithm 1.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the same completion time $T(G_u, n_a) = t_u, \forall G_u \in \vec{G}_u, \forall n_a > 1$, then $T_C^u < T_I^u$, where T_C^u is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_u$ and T_I^u is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_u$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. In $\mathcal{S}_I$, each agent tries completing the same goal simultaneously. First, after completing a goal, the agents switch to the following goal. Thus, at time $T = t_u$ only one goal is completed. Then, at time $T = 2t_u$, two goals are completed. The total time for $\mathcal{S}_I$ to complete all goals is

$$T_I^u = n_G t_u. \tag{B.1}$$

$\mathcal{S}_C$ allows only one agent to work towards each task simultaneously. When a goal is reserved, the agents switch to the next goal. Thus, at time $t_u n_a$, goals have been completed. Then, at time $2t_u$, $2n_a$ goals have been completed. Hence the total time for $\mathcal{S}_C$ to complete all goals is

$$T_C^u = \lceil \frac{n_G}{n_a} \rceil t_u. \tag{B.2}$$

Here, $\lceil \frac{n_G}{n_a} \rceil$ indicates a ceiling division [1].
To prove that $T_C^u < T_I^u$, we will prove that $T_I^u - T_C^u > 0$.
In our case

$$\begin{aligned} T_I^u - T_C^u &= n_G t_u - \lceil \frac{n_G}{n_a} \rceil t_u \\ &= (n_G - \lceil \frac{n_G}{n_a} \rceil) t_u \\ &= \lceil \frac{n_G n_a - n_G}{n_a} \rceil t_u \\ &= \lceil n_G \frac{n_a - 1}{n_a} \rceil t_u . \end{aligned} \tag{B.3}$$

It is clear that $\lceil n_G \frac{n_a - 1}{n_a} \rceil t_u > 0, \forall n_G > 1, \forall n_a > 1$, thus $T_C^u < T_I^u, \forall n_G > 1, \forall n_a > 1$. □

Proving that a system, $\mathcal{S}_C$, using the collaborative extension (Algorithm 2) performs better than a system, $\mathcal{S}_I$, using traditional backward chaining (Algorithm 1) for sets of unshareable goals with different completion times $T(G_u)$ is trickier than when the goals have the same completion time. Note that we can write $T(G_u)$ instead of $T(G_u, n_a)$. Since the goals are unshareable, the completion time is independent of n_a, or $T(G_u, n_a) = T(G_u)$.

B.1.2 Dynamic goal completion time

Here follows a proof that Algorithm 2 gives better completion times than Algorithm 1 for any set of unshareable goals (not necessarily constant goal completion time). The proof is done in two parts.

First, we will prove the statement for the case when there are more agents than goals ($n_a \geq n_G$). Second, we will prove it for the case when there are more goals than agents ($n_a < n_G$). The proof for the case of more agents than goals is straightforward. However, analytical estimations are complex for the case where there are more goals than agents. Then, we prove it by adding a system (Definition B.1), an intermediate solution between Algorithm 1 and Algorithm 2 designed to be straightforward to compare with both.

More agents than goals

Here follows a proof that Algorithm 2 gives better completion times than Algorithm 1 for any set of unshareable goals, when the number of agents is larger than or equal to the number of goals.

Lemma B.2. *When there are more agents than goals, given a set of unshareable goals, the completion time for agents using Algorithm 2 is shorter than for agents using Algorithm 1.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the completion times $T(G_u, n_a) = T(G_u), \forall n_a \geq n_G$, then $T_C^u < T_I^u$. Here T_C^u is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_u$ and T_I^u is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_u$. $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. Looking at the proof of Theorem B.1, we know that $\mathcal{S}_I$ completes each goal one by one. Then the total completion time of $\vec{G}_u$ is

$$T_I^u = \sum_{G_u}^{\vec{G}_u} T(G_u, n_a) = \sum_{G_u}^{\vec{G}_u} T(G_u). \tag{B.4}$$

$\mathcal{S}_C$, on the other hand, allows only one agent to work toward every task simultaneously. Thus, if $n_a \geq n_G$,

$$T_C^u = \max_{G_u \in \vec{G_u}} (T(G_u)). \tag{B.5}$$

If we define G_{umax} as the goal with the longest completion time, or

$$T(G_{\text{umax}}) = \max_{G_u \in \vec{G_u}} (T(G_u)), \tag{B.6}$$

Then

$$T_I^u - T_C^U = \sum_{G_u}^{\vec{G}_u \setminus \{G_{\text{umax}}\}} T(G_u) > 0. \tag{B.7}$$

Hitherto, $T_C^u < T_I^u$ when $n_a \geq n_G$. □

[1] In programming lingo, $\lceil \frac{n_G}{n_a} \rceil = \text{ceil}(\frac{n_G}{n_a})$.

More goals than agents

When there are more goals than agents $n_G > n_a$, it is analytically complex to prove that $\mathcal{S}_C$ performs better than $\mathcal{S}_C$, so it is easier to use a third intermediate system, $\mathcal{S}_D$. Calculating the completion time for $\mathcal{S}_C$ mimics a scheduling problem with no easily expressible analytical solutions. Because of the complexity of the scheduling problem, we construct $\mathcal{S}_D$. $\mathcal{S}_D$ is an alternative to $\mathcal{S}_C$ which also uses 1-to-1 task distribution. However, contrarily to $\mathcal{S}_C$ where the agents reserve new tasks as soon as they complete their previous ones, in $\mathcal{S}_D$ the agents reserve new tasks first when all agents have completed their assigned tasks.

Definition B.1. $\mathcal{S}_D$ is a system that uses collaborative backward chaining as defined in Algorithm 1, but with an additional restriction. The restriction is that all agents in $\mathcal{S}_D$ reserve tasks simultaneously. The agents reserve tasks when no agents in $\mathcal{S}_D$ have an uncompleted reserved task.

Lemma B.3. *When there are more goals than agents, given a set of unshareable goals, a system $\mathcal{S}_D$, where all agents reserve tasks simultaneously, is less efficient than a system $\mathcal{S}_C$, where its agents reserve tasks as soon as the agent becomes available. Both systems use Algorithm 2 where $\mathcal{S}_C$ applies it out-of-the-box and $\mathcal{S}_D$ uses the alteration defined in Definition B.1.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the completion times $T(G_u, n_a) = T(G_u), \forall n_a \in (1, n_G)$, then $T_C^u \leq T_D^u$. Here T_C^u is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_u$ and T_D^u is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_u$. $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 1), and $\mathcal{S}_D$ is a system that uses a deliberately less efficient version of collaborative backward chaining (Definition B.1).

Proof. For both systems, $\mathcal{S}_C$ and $\mathcal{S}_D$, the agents initiates with reserving the first n_a tasks in $\vec{G}_u$. Let $\vec{G}_u^a$ be the subset of $\vec{G}$ containing only the first n_a tasks.

The first task in $\vec{G}_a$ will be completed at

$$T(G^a_{\text{min}}) = \min_{G^a_u \in \vec{G}^a_u} (T(G^a_u). \tag{B.8}$$

Here G^a_{min} is the goal with the lowest completion time.

In $\mathcal{S}_C$, when an agent completes G^a_{min}, it checks the blackboard. The agent observes that the first n_a goals are completed or reserved by other agents and that the first available goal is $(n_a + 1)$:th goal, which the agent reserves. The same thing happens next time an agent becomes available. An agent becomes available when an agent completes the second shortest task or when the first agent has completed two tasks, whichever happens first. Then, again the available agent observes the blackboard. The agent sees that the first $n_a + 1$ goals are completed or reserved and begins working towards completing the $(n_a + 2)$:th task.

The only difference between the two systems is that every agent except the slowest will idle while waiting for other agents to complete their tasks. So, in $\mathcal{S}_D$, when G^a_{min} is done, the assigned agent checks the blackboard and sees that other agents are working towards other tasks. Because at least one agent is still working towards a task, the agent deliberately idles until all agents have completed their assigned task. All agents, except the one assigned to the task with the longest completion time, do the same. The $(n_a + 1)$:th goal is reserved first when the slowest goal is completed, at

$$T(G^a_{\text{max}}) = \max_{G^a_u \in \vec{G}^a_u} (T(G^a_u). \tag{B.9}$$

All agents will idle except the one assigned to G^a_{max}. Limiting ourselves to examining the agent assigned to G^a_{min}. It idles for

$$\max_{i<n_a} (T(\vec{G}(i)) - \min_{i<n_a} (T(\vec{G}(I)). \tag{B.10}$$

$\mathcal{S}_D$ completes all the goals as fast as $\mathcal{S}_C$ except the additional idling. This means that the lower limit of T_D^U is T_C^U, or,

$$T_C^u \leq T_D^U. \tag{B.11}$$

□

Our goal is to prove that for a set of unshareable goals when there are more goals than agents, $\mathcal{S}_C$ (Algorithm 2) has a shorter completion time than $\mathcal{S}_I$ (Algorithm 1). We constructed an intentionally worse rendition of $\mathcal{S}_C$, which we called $\mathcal{S}_D$ (Definition B.1), and for posterity, we also proved that $\mathcal{S}_D$ is less efficient than $\mathcal{S}_C$. The subsequent step is to prove that $\mathcal{S}_I$ is less efficient than $\mathcal{S}_D$, then we can conclude that $\mathcal{S}_I$ is less efficient than $\mathcal{S}_C$.

Lemma B.4. *When there are more goals than agents, given a set of unshareable goals, agents using the weakened version of Algorithm 2 from Definition B.1 are more efficient than agents using Algorithm 1.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the completion times $T(G_u, n_a) = T(G_u), \forall n_a \geq n_G$ (note that the independence from n_a is because of the unshareable property of the goals). Then $T_D^u < T_I^u$, where T_D^u is the time it takes for $\mathcal{S}_D$ to complete $\vec{G}_u$, and T_I^u is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_u$. Here $\mathcal{S}_D$ is a system that uses a less efficient version of collaborative backward chaining (Definition B.1), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. Looking at the proof of Theorem B.1, we know that $\mathcal{S}_I$ completes each goal is completed one by one, which gives us the total completion time,

$$T_I^u = \sum^{n_G} T(G_u, n_a). \tag{B.12}$$

$\mathcal{S}_D$ was constructed and utilized instead of $\mathcal{S}_C$ due to its ability to express the completion time analytically. The agents in $\mathcal{S}_D$ complete the tasks cyclically, where each cycle starts with all agents trying to reserve a task and ends when all agents have completed their task. The agents complete n_a tasks during each cycle except the final one. In the final cycle, unless n_G is evenly divisible with n_a, there are not enough tasks left for 1-to-1 task distribution. There will be two groups of agents. The first group consists of agents who can reserve a task and try to complete it. The second group consists of agents who could not reserve a task and instead idle and wait for the active agents. The number of tasks in the final cycle is rem(n_G, n_a), where rem(a, b) is the remainder of a/b [1].

[1] Many programming languages express the remainder as a % b.

The number of cycles that it takes $\mathcal{S}_D$ to complete all goals in $\vec{G}$ can be defined

$$M = \lceil \frac{n_G}{n_a} \rceil. \tag{B.13}$$

Let us define $\vec{G}_m$ as the subset of the goals completed in the $m:th$ cycle. Note that

$$\{\vec{G}_m\}_m^M = \vec{G}_u. \tag{B.14}$$

Then, the time it takes for the agents in $\mathcal{S}_D$ to complete all the goals in $\vec{G}_u$ can be expressed as

$$T_D^u = \sum_m^M T(\vec{G}_m). = \sum_k^{\lceil n_G/n_a \rceil} T(\vec{G}_m), \tag{B.15}$$

where $T(\vec{G}_m)$ is the time it takes to complete the subset $\vec{G}_m$.

We know that a cycle ends and the next cycle starts, first when all agents have completed their assigned tasks. If we denote $G_{m,\max}$ as the goal that takes the longest time to complete in the subset $\vec{G}_m$, then $T(\vec{G}_m) = T(G_{m,\max})$. Given that $\vec{G}_{\max}$ is the set of goals $G_{m,\max}{}_m^M$, it follows that Equation (B.15) also can be written as

$$T_D^u = \sum_k^{\lceil n_G/n_a \rceil} T(\vec{G}_m) = \sum_k^{\lceil n_G/n_a \rceil} T(G_{m,\max}) = T(\vec{G}_{\max}). \tag{B.16}$$

Equation (B.12) and Equation (B.16) are sufficient to show that $T_D^u < T_I^u$. We do this by demonstrating that $T_I^u - T_D^u > 0$. Using Equation (B.12) and Equation (B.16) we can express this difference as

$$T_I^u - T_D^U = \sum_{G_u}^{\vec{G}_u \setminus \vec{G}_{\max}} T(G_u) > 0. \tag{B.17}$$

□

Theorem B.5. *Given a set of unshareable goals, the completion time for agents using Algorithm 2 is shorter than for agents using Algorithm 1.*

Assume there is a set of unshareable goals $\vec{G}^u$ with n_G goals, where $n_G > 1$ with the completion times $T(G_u, n_a) = T(G_u), \forall n_a > 1$, then $T_C^u < T_I^u$, where T_C^u is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_u$ and T_I^u is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_u$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. To prove the theorem that agents using Algorithm 2 are more efficient in completing unshareable goals than agents using Algorithm 1, we can conveniently utilize Lemma B.2, Lemma B.3 and Lemma B.4 together. Lemma B.2 proves that Algorithm 2 is more efficient when there are more agents than goals. Meanwhile, Lemma B.3 and Lemma B.4 prove the same thing when there are more goals than agents.

Lemma B.3 shows that $T_C^u \leq T_D^u$ for $n_a < n_G$. Lemma B.3 shows that $T_D^u < T_I^u$ for $n_a < n_G$. Thus $T_C^u < T_I^u$ for $n_a < n_G$. Lemma B.2 shows that $T_C^u < T_I^u$ for $n_a \geq n_G$. Thus finally, $T_C^u < T_I^u, \forall n_a > 1, \forall n_G > 1$. □

B.2 Shareable goals

In this section, we prove that for shareable goals (Definition 5.1), with no goal-switching costs (Definition 5.4), traditional backward chaining (Algorithm 1) performs equally well or better than collaborative backward chaining (Algorithm 2).

Like with the proofs regarding unshareable goals, the ones regarding shareable goals are done by first looking at constant completion time and then at a dynamic completion time. Additionally, as for shareable goals, the proof is separated by two cases; whether there are more goals than agents or there are more agents than goals.

Additionally, note that our proofs are limited to homogenously shareable goals (see Definition 5.5).

B.2.1 Constant goal completion times

Theorem B.6. *Given a set of shareable goals with the same completion time, agents using Algorithm 1 has a shorter completion time than agents using Algorithm 2.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with the same completion time $T(G_s, n_a) = t_s/n_a, \forall G_s \in \vec{G}_s, \forall n_a > 1$, then $T_C^s \geq T_I^s$, where T_C^s is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_s$ and T_I^s is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_s$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. Let us look at a simplified scenario limited to a system trying to complete a single homogenously shareable goal G_s.

$\mathcal{S}_I$ has all n_a agents working in parallel to achieve G_s in

$$t_I^s = \frac{t_s}{n_a}. \tag{B.18}$$

In $\mathcal{S}_C$, on the other hand, only one agent works simultaneously on the task on the task. While the allocated agent actively works towards the goal, the other $n_a - 1$ agents idle. Thus, the system completes the goal in

$$t_C^s = t_s. \tag{B.19}$$

This idling agent issue remains for several goals as long as $n_G < n_a$.
Since the goals are homogenously shareable, $\mathcal{S}_I$ achieve the goals $\vec{G}_s$ in

$$T_I^s = \frac{n_G}{n_a} t_s. \tag{B.20}$$

However, in $\mathcal{S}_C$, the agents end up in two groups. The first group contains n_G agents that reserve a goal and tries to complete it. The second group holds $n_a - n_G$ agents that have no available goal that they can reserve, and therefore they instead idle. $\mathcal{S}_C$ then requires that

$$T_C^s = \lceil \frac{n_G}{n_a} \rceil t_s. \tag{B.21}$$

Equation (B.20) and Equation (B.21) are enough to show that $T_I^s < T_C^s$. We do this by demonstrating that $T_C^s - T_I^s > 0$,

$$T_C^s - T_I^s = (\lceil \frac{n_G}{n_a} \rceil - \frac{n_G}{n_a}) t_s = (1 - \text{rem}(n_G, n_a)) t_s \geq 0. \tag{B.22}$$

Here rem(a, b) is the remainder of $\frac{a}{b}$. $\square$

B.2.2 Dynamic goal completion times

Next, we can increase the complexity and prove that agents using Algorithm 1 have a shorter completion time than agents using Algorithm 2 for a set of homogenously shareable goals with different completion times. As for shareable goals, it is easiest to separate the proof into two parts. The case when there are more goals than agents ($n_a \geq n_G$) and the case when there are more agents than goals ($n_a < n_G$).

More agents than goals

The following lemma shows that Algorithm 1 gives better completion times than Algorithm 2 for any set of homogenously shareable goals when the number of agents is larger than or equal to the number of goals.

Lemma B.7. *When there are more agents than goals, given a set of shareable goals, Algorithm 1 has a shorter completion time than agents using Algorithm 2.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with completion times

$$T(G_s, n_a) = \frac{T(G_s, 1)}{n_a} = \frac{T(G_s)}{n_a}, \forall G_s \in \vec{G}_s, \forall n_a \geq n_G. \tag{B.23}$$

Then $T_C^s \geq T_I^s$, where T_C^s is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_s$ and T_I^s is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_s$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. We limit ourselves to when there are more agents than goals ($n_a \geq n_G$).
In $\mathcal{S}_I$ all n_a agents works in parallel to achieve each goal G_s. We know from Equation (B.18) that each goal is completed in

$$t_I^s = \frac{T(G_s)}{n_a}. \tag{B.24}$$

We also know that the agents complete the goals in order. This means that the total time for $\mathcal{S}_I$ to complete all goals in $\vec{G}_s$ is

$$T_I(\vec{G}_s) = \sum_{G_s}^{\vec{G}_s} \frac{T(G_s)}{n_a}. \tag{B.25}$$

$\mathcal{S}_C$, on the other hand, uses 1-to-1 mapping so that n_G agents work towards completing their own 1-to-1 assigned goal and $n_a - n_G$ agents idle. Using equation (B.19) we know that each goal is completed in

$$t_C^s = \frac{T(G_s)}{1} = T(G_s). \tag{B.26}$$

However, since the agents in $\mathcal{S}_C$ operate in parallel, the agents have completed the goals $\vec{G}_s$, when an agent has completed G_{smax}. Here G_{smax} is the goal that takes the longest to complete. G_{smax} is defined by

$$T(G_{\text{smax}}) = \max_{G_u \in \vec{G}_u} (T(G_u)). \tag{B.27}$$

With G_{smax}, the completion time for $\mathcal{S}_C$ can be written as

$$T_C^s = T(G_{\text{smax}}) = n_a \frac{T(G_{\text{smax}})}{n_a}. \tag{B.28}$$

Inspired by this, we look at Equation (B.25) to reveal an upper bound for the completion time for $\mathcal{S}_I$,

$$T_I^s = \sum_{G_s}^{\vec{G}_s} \frac{T(G_s)}{n_a} \leq \sum_{G_s}^{\vec{G}_s} \frac{T(G_{\text{smax}})}{n_a} = n_G \frac{T(G_{\text{smax}})}{n_a}. \tag{B.29}$$

We will show that $T_I^s < T_C^s$ using (B.29) and Equation (B.28). It follows from those equations that $T_I^s < T_C^s$ is equivalent

$$n_g \frac{T(G_{\text{smax}})}{n_a} \leq n_a \frac{T(G_{\text{smax}})}{n_a}. \tag{B.30}$$

Thus we have proven that $T_C^s \geq T_I^s$. We know that $\frac{T(G_{\text{smax}})}{n_a} \neq 0$, we can divide each side of the comparison with that factor and see directly $T_C^s \geq T_I^s$ for $n_g \leq n_a$. □

More goals than agents

The next step is to look at the proof that agents using Algorithm 1 are more efficient than agents using Algorithm 2 for a set of shareable goals when there are more goals than agents ($n_a < n_G$). This proof is slightly more abstract as it utilizes the scheduling properties of Algorithm 2.

Lemma B.8. *When there are more goals than agents, given a set of shareable goals, Algorithm 1 has a shorter completion time than agents using Algorithm 2.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with completion times $T(G_s, n_a) = T(G_s, 1)/n_a = T(G_s)/n_a, \forall G_s \in \vec{G}_s, \forall n_a \in (1, n_G)$, then $T_C^s \geq T_I^s$, where T_C^s is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_s$ and T_I^s is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_s$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. As in the case for $n_a \geq n_G$, the completion time for $\mathcal{S}_I$ is

$$T_I^s = \sum_{G_s}^{\vec{G}_s} \frac{T(G_s)}{n_a}. \tag{B.31}$$

The one for $\mathcal{S}_C$ is trickier. Let us define $\vec{a}$ as the agents and $\vec{g}(a)$ as the subset of goals $\vec{G}$ that the agent $a \in \vec{a}$ will work on. Given these properties, the total time is

$$T_C^s = \max_{a \in \vec{a}} \sum_{G}^{\vec{g}(a)} = \max_{a \in \vec{a}} T(\vec{g}(a)). \tag{B.32}$$

What defines $\vec{g}(a)$? Each time an agent completes the goal, the next goal is added to the agent. This property makes T_C^s analytically complex to express for a generic set of $\vec{G}$. However, we can find a lower bound for T_C^s by minimizing T_C^s, given $\vec{G}$. T_C^s is minimized when $T(\vec{g}_a)$ is constant in $\vec{a}$, which can be expressed as

$$T_C^s = T(\vec{g}(a)). \tag{B.33}$$

We also know that the total time of all agent's goals is

$$\sum_{a}^{\vec{a}} T(\vec{g}(a)) = \sum^{n_G} T(G_s). \tag{B.34}$$

If $T(\vec{g}(a))$ is constant in $\vec{a}$ this is equivalent to

$$n_a T(\vec{g}(a)) = \sum_{G_s}^{\vec{G}_s} T(G_s). \tag{B.35}$$

Hence, the minimum value, or lower bound, of T_C^s is

$$T_C^s \geq \frac{1}{n_a} \sum_{G_s}^{\vec{G}_s} T(G_s) = \sum_{G_s}^{\vec{G}_s} \frac{T(G_s)}{n_a} = T_I^s, \tag{B.36}$$

or $T_C^s \geq T_I^s$.

□

Theorem B.9. *Given a set of shareable goals, agents using Algorithm 1 has a shorter completion time than agents using Algorithm 2.*

Assume there is a set of homogenously shareable goals $\vec{G}$, defined in Definition 5.5, where $n_G > 1$ with completion times $T(G_s, n_a) = T(G_s, 1)/n_a = T(G_s)/n_a, \forall G_s \in \vec{G}_s, \forall n_a > 1$, then $T_C^s \geq T_I^s$, where T_C^s is the time it takes for $\mathcal{S}_C$ to complete $\vec{G}_s$ and T_I^s is the time it takes for $\mathcal{S}_I$ to complete $\vec{G}_s$. Here $\mathcal{S}_C$ is a system that uses collaborative backward chaining (Algorithm 2), and $\mathcal{S}_I$ is a system that uses traditional backward chaining (Algorithm 1).

Proof. All the relations between the number of goals and the number of agents relationships are covered by Lemma B.7 and B.8 together. Lemma B.7 shows that $T_C^s \geq T_I^s$ when there are fewer or as many agents than goals ($n_a \leq n_G$). Lemma B.8 shows that $T_C^s \geq T_I^s$ when there are more agents than goals ($n_a > n_G$).

Lemma B.7 and B.8 together shows that

$$T_C^s > T_I^s, \forall n_a > 1, \forall n_G > 1. \tag{B.37}$$

□