UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS700 Research Report

Development and Implementation of a Framework for Synthesising Strategy Profiles for Multi-Robot Systems

Student number: u18053239

Supervisor(s):
Dr. Nils Timm

Arul Agrawal

October 2023

Abstract

Robots moving around in a warehouse to pickup and deliver items can be regarded as competitive scenario of accomplishing tasks in multi-agent systems. Typically, there are overall global objectives of the system, as well as individual objectives of each robot. Objectives can be classified as qualitative objectives that can be expressed in temporal logic and quantitative objectives such as maximising the frequency of accomplishing tasks. This project aims to design and implement a framework which can be used to synthesize a strategy profile for the robots which fully or at least approximately satisfies various requirements of the system.

Keywords:

Multi-agent systems, strategy, robots, Epsilon-equilibrium, step-by-step optimisation, warehouse delivery, spin

1 Introduction

Multi-agent systems (MAS) are systems composed of multiple autonomous agents that interact in a shared environment. Each agent with its own attributes, capabilities and decision-making collectively contributes to the system's ability to achieve its overall objectives. Each agent also has individual objectives, which it must try to fulfill while simultaneously fulfilling the overall system objectives.

Each agent has to make a decision (i.e. perform an action) when the system is in a given state. The sum of all these decisions, or a decision for each state in the system, forms the strategy for that agent. The strategy for all the agents comprises a strategy profile for the system.

Each agent has associated qualitative objectives, expressed in linear temporal logic (LTL) [1], such as safety or fairness, as well as quantitative objectives, like to maximize some payoff. This project aims to use a model which combines qualitative and quantitative objectives. Like in [2] we consider a system where an agent's primary objective is to satisfy its qualitative objectives and an agent's secondary objective is to maximise its payoff (which will be defined by some payoff function). This approach has the advantage that the qualitative objectives can be used to indicate desirable properties of the system, as is standard when using LTL. The quantitative objective can be used the restrict the strategies of the agents to those strategies which are optimal for each agent.

Thus we can design a combination of objectives which together state that we want an agent to accomplish its qualitative objective, but also do so as efficiently or quickly as possible. However, agents may have conflicting objectives such that a strategy profile which is optimal for each agent may not exist. Hence, it is necessary to define properties which can be fully or at least approximately be satisfied by a given strategy profile. An additional challenge is that if even properties have been identified and strategy profiles exist, the synthesis of these may be computationally too expensive.

In this project we consider a multi-agent system which models the floor of an automated warehouse. In the Warehouse Game (WG), a number of robots move around in a grid to load items and deliver them to an exit point. Their primary objective is to load an item and deliver it to the correct exit point. While achieving this primary objective, there are other objectives, like to deliver efficiently (that is to deliver as many items as possible), to avoid crashing into the other robots, or to always make at least one delivery in a given time period. In the context of the warehouse game (and this report) an agent and a robot are synonymous.

We must consider the problems of *deadlock* and *starvation*. A naive

solution would be a greedy strategy - one in which all robots aim to pickup the item which is closest to itself, and then deliver that item to the exit point using the shortest path to the exit. This strategy however leads to *deadlock*, if two robots are trying to move past each other, collect the same item or follow the same path. It also can lead to *starvation*, in the case where a robot happens to always be further away from an item than some other robot, always arriving at the item too late to pick it up.

From the possible problems of *deadlock* and *starvation*, we can derive corresponding qualitative objectives that our strategy needs to satisfy, namely *deadlock*-freedom and *starvation*-freedom.

There may be multiple strategies which avoid *deadlock* and *starvation* and which also fulfill the objectives, however we must be able to decide between them. We might prefer a strategy which is fairer - one in which the workload is more evenly distributed amongst the agents - to one in which a few of the robots are performing the majority of the work. This project aims to develop an approach for automatically synthesizing fair strategy profiles for warehouse games such that:

- If the agents follow the strategy profile, the collective objective is accomplished.
- No agent can deviate from the strategy profile and improve its individual payoff if the remaining agents still follow the strategy profile. This is related to the concept of Nash-equilibria and stability. [1]

The warehouse game is a useful analogue to some real world problems. It models systems in logistics, like manufacturing, warehousing, deliveries and other logistics. It offers a way to compute strategies in advance, which can guarantee desirable properties, like efficiency or safety.

Synthesising a fair equilibrium for the WG is appealing because it ensures a balanced outcome in which each of the robots is doing its fair share of work. However, fair equilibria may or may not exist, and finding a fair equilibrium or proving that one does not exist is a computationally difficult problem because the state space of the system can grow very large. We aim to develop an iterative optimisation technique in order to synthesise fair equilibria. In addition to generating fair equilibria, the concept of total payoff is also considered. In this case, only the total payoff obtained is considered, and a given robot may not do any work if it results in the overall payoff being higher. In this case an individual agent may face *starvation*, but as a whole the system will still be *deadlock* and *starvation* free.

The technique implemented here relies on a novel usage of the SPIN model checker, specifically in a way that leverages SPIN's ability to provide

counterexamples to LTL formulae. We setup our system in a way that a counterexample from SPIN corresponds to a valid strategy profile for the warehouse game.

The rest of the project describes formally the components needed to conceptualise this problem, namely linear temporal logic, strategies and runs. These components are use to define the warehouse game as well as the strategies of the robots. The implementation of the framework to generate these strategies is explained next, followed by some experiments to test the feasibility of the approach employed.

2 Related Work

Multi-agent systems are core part of the model checking field. In this section we lay out some related prior work which has been done in this field, both in terms of the model checking process as well as strategy synthesis.

2.1 Survey on multi-agent systems

[3] features a high-level overview of multi-agent systems. It illustrates the variety of applications of MAS, as well as some of the research challenges which need to be solved in order to make the application of MAS more widespread and useful. It introduces concepts such as localization (where an agent only has a smaller, local view of its surroundings rather than the whole state) and agent organization, where agents may be heterogeneous. It also discusses routing of robots and agents in robotics, which corresponds to the Warehouse game described in our project.

2.2 Challenges and opportunities in pickup and deliver problems

[4] outlines the current challenges and opportunities in what they call Multi-Agent Path Finding (MAPF) and Multi-Agent Pickup and Delivery Problems (MAPD). They explore modelling Amazon robots in a warehouse moving pods which contain goods to drop off locations. They discuss the difficulty in finding efficient paths for MAPF, and the difficulty of deciding whether or not an instance of MAPD is solvable. This project aims to develop a method to generate strategy profiles for MAPDs experimentally.

2.3 Games with combined qualitative and quantitative objectives

[2] introduces the concept of combining qualitative and quantitative objectives, and also introduces an automated warehouse, which serves as the basis for the WG defined in this project. They give it as an example of a game but we will be exploring it in greater depth. The main result of their paper is a proof that an algorithm exists to decide whether or not a game has a finite-state strict ϵ Nash-equilibrium, however they do not synthesise it. Our goal is to use the result that an equilibrium exists, and to synthesise it experimentally using an iterative optimisation technique.

2.4 Reasoning about strategies

Their paper [1] establishes concurrent game structures, and provides rock-papers-scissors and the prisoner's dilemma as examples. Our warehouse games are a special case of the concurrent game structures described in their paper. Further, they introduce strategy logic, an extension of LTL which may become useful in future work aimed at extending the approach in this report. They also introduce the idea of equilibria, and differentiate between unstable and stable equilibria.

2.5 Strategy synthesis for resource allocation

In [5], the authors create an implementation to decide the model checking problem for resource allocation systems and also to synthesise a strategy using a SAT solver over ATL* (alternating time logic) formulas. This approach is different to the one used in this report, as the warehouse game is not a resource allocation problem. Further, this project considers objectives only in terms of LTL, not ATL*, and we use the SPIN model checker directly, instead of converting to boolean satisfiability.

2.6 Synthesis of uniform strategies for MAS

[6] proposes a technique for the synthesis of strategies in systems where agents may have incomplete information, and proposes uniform strategies, where the the same choice is made in 2 states which are different, but indistinguishable based on information available to the agent being considered. They use an approach to limit the search space, and develop a method called SMC (Strategic Model Checker) and describe the algorithm behind it, which uses

an approach involving partial strategies and iteratively fixing actions for unknown states until a valid strategy is found. This approach performed well, and very favourably against the well-known ATL model checker MCMAS [7]. Our project also considers uniform strategies where agents have incomplete information.

2.7 SPIN model checker

The SPIN model checker [8] is another well-known model checker. It deals with LTL formulas, and contains many optimisations for model checking such as partial order reductions. Our report uses the SPIN model checker to generate strategy profiles which are guaranteed to meet the qualitative and quantitative objectives desired. The approach employed in this report is novel, as SPIN's primary purpose is to check systems and strategies, not to generate strategies.

2.8 Max-SAT approach

The approach of synthesising strategies of MAS using Max-SAT [9] is explored in [10]. Though this paper also deals with the resource allocation problem like in [5], it provides an overview for using a Max-SAT approach instead of a SAT approach, since there may exist formulations where complete satisfiability is not possible. It would then be necessary to use a Max-SAT approach to satisfy as many of the clauses as possible. The paper provides an algorithm to encode coalition and opposition strategies, and an algorithm to synthesise collectively optimal and Nash-equilibrium strategies using the Max-SAT approach. SAT and Max-SAT may provide more efficient ways to synthesise strategies for the warehouse game, but were not used in this project.

3 Background

In order to precisely define the warehouse game, and the properties we want our synthesised strategy profile to solve, we must first define concepts and terminology which are commonly used in model checking.

3.1 Atomic propositions, states and paths

Atomic propositions are boolean variables which express information about the system. In the case of the warehouse game, concepts like whether or not an agent has crashed with another agent, if an agent has scored or if an agent is currently loading an item are all examples of atomic propositions. The set of all atomic proposition is referred to as AP.

A state is a complete description of the system at a specific point in time, characterized by the valuations of all atomic propositions. A state captures a snapshot of the system, encapsulating every detail necessary to describe the system's current situation. A state can therefore also be considered as a set of atomic propositions, which represents information which is true in that state.

In the case of the warehouse game, a state is characterised by the locations of the robots, whether or not each robot is carrying an item, as well as few other atomic propositions described in section 4.1.

A path (or run) of the system is then the sequence of states indicating the progression or evolution of the system over time. When considering paths in LTL:

- Paths are infinite sequences of states, reflecting that systems are analyzed over an indefinite time horizon. Since the number of states is finite, an infinite sequence must contain a loop.
- The transition from one state to the next in a path represents a change in the system, which may change the truth values of one or more atomic propositions. In this case a transition indicates that each agent has made a move, and possibly loaded or delivered an item.
- LTL is a temporal logic that can be used to analyse these paths, assessing how the truth values of atomic propositions and thus the state of the system change over time.

We denote a state s, and s_n to indicate the (n+1)th state in a path π. Each path always begins at the initial state s_0. Therefore $\pi = s_0 s_1 s_2 s_3 ...$ A suffix π^i is defined as $\pi^i = s_i s_{i+1} s_{i+2} ...$, i.e. the i-th suffix of π is the (i+1)th state of π and its subsequent states.

3.2 Linear Temporal Logic (LTL)

Linear Temporal Logic is a modal temporal logic with modalities referring to time. In LTL, one can express statements not only about the present

state of a system but also about its future states. It allows for the expression of properties such as safety, fairness and liveness. Below is the inductive formulation of LTL over AP:

$$\phi ::= p \mid \neg\psi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid X\phi \mid F\phi \mid G\phi \mid \phi\, U\psi$$

An LTL formula can be satisfied by a path π. The satisfaction relation $\vDash$ is used $\pi \vDash \phi$, meaning that the path π satisfies the LTL formula ϕ (with $p \in AP$ and ϕ and ψ are LTL formulas):

- $\pi \vDash p$ iff $p \in s_0$
- $\pi \vDash \neg\psi$ iff it is not the case that $\pi \vDash \psi$
- $\pi \vDash \phi \wedge \psi$ iff $\pi \vDash \phi$ and $\pi \vDash \psi$
- $\pi \vDash \phi \vee \psi$ iff $\pi \vDash \phi$ or $\pi \vDash \psi$
- $\pi \vDash \phi \rightarrow \psi$ iff $\pi \vDash \neg\phi$ or $\pi \vDash \psi$
- $\pi \vDash X\phi$ iff $\pi^1 \vDash \phi$
- $\pi \vDash \phi\, U\psi$ iff $\exists k \geq 0(\pi^k \vDash \psi$ and $\forall i < k(\pi^i \vDash \phi))$
- $\pi \vDash F\phi$ iff $\pi \vDash \text{true}\, U\phi$
- $\pi \vDash G\phi$ iff $\pi \vDash \neg F\neg\phi$

Intuitively the main temporal operators (X, F and G) can be understood as:

- X is the next operator. $X\phi$ states that ϕ is satisfied in the next state.
- F is the eventually operator. $F\phi$ states that ϕ is eventually satisfied, the system will reach a state where ϕ is satisfied.
- G is the globally operator. $G\phi$ states that ϕ is always satisfied, meaning that the atomic propositions of ϕ are true in every state.

4 Conceptualisation

In this project we consider a multi-agent system which models the floor of an automated warehouse. Each agent has its own load point and exit point. The objective of each agent is to visit the load point and then the exit point, thus making a delivery. We seek to synthesise a strategy for each agent to facilitate this, i.e. to compute the precise sequence of moves that each agent needs to make in order to make deliveries.

4.1 Operational model of Warehouse Game

We use a modification of the model introduced in [2].

The warehouse itself is represented by a directed edge-labeled graph $G = (V, E, D)$ where $E : V \times D \to V$ for a finite set of directions D. In this case, $D = \{north, south, east, west\}$ creating a 2 dimensional grid. An agent at a vertex v taking the action $d \in D$ will move to the position given by $E(v, d)$.

A Warehouse game is the tuple:

$$WG = \langle G, Agt, Act, St, \iota, \tau, LP, EP \rangle$$

where:

- G is the directed edge-labeled graph.
- $Agt = \{1, ..., n\}$ is the non-empty set of agents
- $Act = D$ is the list of actions each agent can take
- $St = Agt \times V \times \mathbb{B}$, where $\mathbb{B}$ is the set of Booleans. St represents the set of all states, where each state $s : Agt \to V \times \mathbb{B}$ is a function that maps each agent to a tuple containing that agents position, and a Boolean indicating whether or not it is carrying an item.
- $\iota \in St$ is the initial state, denoting the initial positions of the agents. It is assumed that the agents are not carrying an item in the initial state.
- $\tau : St \times Act^{|Agt|} \to St$ is a transition function that maps a state and the actions of each agent to its successor state.
- $\forall a \in Agt : LP_a \in V$ is the loading point for agent a.
- $\forall a \in Agt : EP_a \in V$ is the exit point for agent a. Additionally it is assumed that $\forall a \in Agt : LP_a \neq EP_a$.

We introduce some shorthand notation to express relevant information. For a state $s \in St$ and agent $a \in Agt$, $s(a) = (v, b)$ where $v \in V$ is the position of agent a in state s, and $b \in \mathbb{B}$ is the Boolean indicating whether or not the agent is carrying an item in this state. Additionally the subscript notation $s(a)_v$ and $s(a)_b$ denotes just the v or b component respectively of the tuple explained above.

We also introduce some atomic propositions (AP) as a short way of expressing certain states with certain properties [2]. For each agent $a \in Agt$ and a state s:

- $load_a \in s \equiv (s(a)_v = LP_a)$
- $exit_a \in s \equiv (s(a)_v = EP_a)$
- $crash_a \in s \equiv (\exists\, b \neq a : s(a)_v = s(b)_v)$

An agent automatically picks up an item when it lands on a loading point: $load_a \rightarrow s(a)_b = true$ and automatically completes the delivery when it lands on an exit point while carrying an item: $exit_a \wedge s(a)_b$.

In this operational model we also assume that the agents can move past each other - at a state s if $s(a)_v = v_x$ and $s(b)_v = v_y$ where $v_x, v_y \in V$, then at the successor state s_1 it is possible that $s_1(a)_v = v_y$ and $s_1(b) = v_x$, i.e. the agents have "swapped" places. This corresponds to the intuitive idea that in the warehouse the robots have the space to move past one other. Alternative models are proposed in section 8.

4.2 Strategy

The strategies synthesised in the paper are uniform and local. A uniform strategy is one where an agent always makes the same decision when it encounters the same state. A strategy being local means that each agent only has knowledge of its own attributes, i.e. its own position and whether not it is carrying an item.

This means that each agent a, has a strategy function $\eta_a : V \times \mathbb{B} \rightarrow Act$. In a strategy η_a, the input tuple (v, b) (the position of the agent, and whether or not it is carrying an item) determines the action that it takes. Whenever the agent is in the state (v, b), it will always take the same action.

4.3 Objectives

Each agent has qualitative objectives which it aims to fulfill first, and then secondary quantitative objectives.

4.3.1 Qualitative objectives

The most important objective of the strategies generated is safety, which in this case means that the agents never "crash", i.e. no 2 agents can occupy the same vertex in a given state. Each agent a has the LTL objective $safety_a = G(\neg crash_a)$. Therefore our overall safety goal is:

$$G(safety_a)\ \forall a \in Agt \tag{1}$$

The requirement that an individual agent makes progress can also be expressed as a qualitative objective:

- $deliverOnce_a = F(load_a) \wedge F(load_a) \rightarrow F(exit_a)$. The requirement of an agent to eventually load and deliver.
- $deliver_a = G(F(load_a)) \wedge G(F(load_a) \rightarrow F(exit_a))$. This is a *liveness* requirement, stating that each agent must infinitely often load and deliver an item.

In general $deliver_a$ is a much stronger requirement than $deliverOnce_a$, since it implies that an agent must deliver infinitely often, as opposed to just once. We however express the progress objectives as quantitative objectives, explained below.

4.3.2 Quantitative objectives

The quantitative objectives for an agent will be defined by a payoff function. We can define a simple payoff function for an agent a as $P_a : St \rightarrow \mathbb{Z}$. It is natural to reward an agent for delivering an item, as this is the objective of the system - for a state $s \in St$, $P_a(s) = 1$ if $s(a)_v = EP_a \wedge s(a)_b = true$ and otherwise 0.

We measure how well an agent performs under a certain strategy profile by limiting the game to number of steps k. After k steps, the game is simply stopped, an agent a's score T_a is the sum of its payoffs in the k states along its path, where a higher score indicates a better performance. The mean-payoff in this case is $mp(a) = \frac{T_a}{k}$, the ratio of payoff to steps.

Since the set of states St is finite, and there exists a successor state $\tau(s, d)$ for $s \in St$ and $d \in Act^{|Agt|}$ for every state, it is guaranteed that a path π will eventually enter a loop: $\pi = s_0 s_1 s_2 ... s_n$. If the successor of a state s_n is a previously encountered state s_l, $l < n$, then the path has entered an infinite loop from l to n due to the uniform nature of the strategies.

This means that we can model the infinite behaviour of the path in a finite number of steps, as long as we choose our number of steps $k > n$, which ensures that the loop is captured within in the steps. The value of k necessary depends on the scenario being considered, simple scenarios where a loop is entered quickly only require a small k, and complex scenarios with agents taking long paths will require larger values of k.

We consider two main quantitative objectives. The first and simplest is

total mean-payoff, which is sum of the payoffs of each agent:

$$MP = \sum_{a=1}^{n} mp(a)$$

where n is number of agents. As quantitative goal, we simply state

$$F(MP \geq \gamma) \tag{2}$$

where γ is minimum total mean-payoff we want the path generated by the strategy profile to have.

This goal can lead to unbalanced strategies; since we only care about the total, it is possible that one agent is doing the majority (or possibly all) of the work, while the other agents are not doing any work (i.e. $mp(a) = 0$ for some agent a). In order to avoid this, we can instead use mean-payoff to introduce fairness constraints. If we give each agent a a payoff goal γ_a, then we can ensure that each agent achieves a mean-payoff of γ_a by using the LTL formula:

$$F(mp(a) \geq \gamma_a) \, \forall a \in Agt \tag{3}$$

By setting each $\gamma_a = 1/k$, we can ensure that each agent delivers at least one item in k steps. This is equivalent to the $deliverOnce_a$ requirement given above. The agent a will enter a loop, but it is not guaranteed that this loop is one where the agent a scores. Thus it is possible that the agent a scores once, and only once. This means that in order to enforce the *liveness* property $deliver_a$, we must set the target $\gamma_a \geq 2/k$, keeping in mind the requirements of choosing a suitable k. $2/k$ is sufficient since it is impossible to score twice without entering a loop - as soon as the agent lands on the load point again, it has entered a loop.

It is important to note that we must be careful in our choice of γ and γ_a. If the values chosen are too high, then there will not be any strategy which meets the requirements. Imagine a simple scenario where there is only 1 agent a. If $load_a \in s_l$ and $exit_a \in s_e$, and s_l and s_e are 2 steps away from each other (in terms of the path), then the shortest possible loop $s_l s_{l+1} s_{l+2} s_e s_{l+2} s_{l+1} s_l ...$ has a length of 6 (the loop begins again on the 7th step). This means that the highest possible mean-payoff for the agent is $\frac{1}{6}$. An attempt to find a higher mean-payoff will always result in failure because the shortest path is already being taken - the strategy generated is already optimal.

5 Implementation

We go over the approach used the synthesise strategy profiles, by leveraging the SPIN model checker, as well as writing a custom tool to feed scenarios to spin, and parse its output. A GUI visualiser is then generated from the strategy information produced by SPIN.

5.1 SPIN Model Checker

The implementation for model checking and strategy synthesis relies primarily on the SPIN model checker [8]. SPIN is a well-known model checker, developed by Bell Labs in the 1980s. Systems (like our warehouse game) are modelled in SPIN using its language Promela, and properties of the system (our objectives) are expressed in LTL. SPIN negates these LTL formulas and converts them to Büchi automata as part of the model checking process. Crucially, SPIN contains many optimizations for model checking, such as converting the program to C, as well as state space optimisations.

The SPIN model checker is usually used to verify system properties of a system. SPIN has been widely used for this purpose. We however use SPIN in the "opposite" way - we give SPIN the system, but ask it to generate strategies by making random choices, and prune away undesirable strategies using our objectives described in LTL. This is possible due to a core feature of SPIN, the trail. During normal execution, if SPIN encounters an error, i.e. the system does not conform to the desired properties, it reports this error, as well as the full execution trail - a so-called counterexample path. This lets us examine the exact scenarios in which our system fails to satisfy the LTL.

In our case, we negate the objectives (eqs. (1) to (3)) first, then pass them to SPIN along with the algorithms below. When SPIN reports an error and a execution trail, it will contain all the random choices made, thus comprising a full strategy profile. Since we negated the objective goals, the trail returned does not represent an error, but actually represents a valid strategy profile in which all the objectives are satisfied.

We implement a slightly simplified version of the warehouse game, namely that instead of any directed edge-labelled graph, we implement only a 2D grid, with moves possible in the 4 cardinal directions. The grid has dimensions m by n, which is an input parameter of the system. The value k is the one chosen in 4.3.2.

The process that models the possible behaviour of an agent is described by the algorithm below:

An agent does not handle its own movement, loading and delivering or scoring directly. This task is instead delegated to a special environment

Algorithm 1 Strategy Synthesis for an agent

$i \leftarrow 0$
$N \leftarrow k$ ▷ The number of steps to simulate
$x \leftarrow x$ ▷ The initial position is set directly
$y \leftarrow y$
$moves \leftarrow$ an empty map
$move \leftarrow none$ ▷ The move chosen for the current state
while $i \neq N$ **do**
    $loaded \leftarrow$ **true** *or* **false** ▷ Whether or not the robot has an item
    $index \leftarrow m * y + x$
    if $moves[index][loaded]$ is known **then** ▷ known state
        $move \leftarrow moves[index][loaded]$
    else if $moves[index][loaded]$ is unknown **then** ▷ new state
        $move \leftarrow$ a valid random move ▷ a move that stays within bounds
        $moves[index][loaded] \leftarrow move$
    end if
    $i \leftarrow i + 1$
end while

process which handles these for all agents. The qualitative safety objective (to never crash) and the quantitative scoring objectives are described with Ensure clauses:

Algorithm 2 The environment

Ensure: no crash ▷ Objective 1
Ensure: $score > \gamma$ ▷ Objective 2 or 3
  $i \leftarrow 0$
  $N \leftarrow k$ ▷ The number of steps to simulate
  $Agt \leftarrow 1, 2, ...$ ▷ The set of agents
  while $i \neq N$ **do**
    for all $agent \in Agt$ **do**
      $move \leftarrow$ move from agent
      $position$ is updated ▷ each agent has its own position
      if $load_{agent}$ **then** ▷ agent is in the load state
        $loaded_{agent} \leftarrow$ **true** ▷ agent is now has an item
      else if $exit_{agent}$ **and** $loaded_{agent}$ **then** ▷ agent makes delivery
        $score \leftarrow score + 1$
        $loaded_{agent} \leftarrow$ **false**
      end if
    end for
    $i \leftarrow i + 1$
  end while

If the execution reaches a state where crash occurs, or if after k steps the quantitative objectives are not met, that execution path is considered an invalid strategy, and is not returned by SPIN.

5.2 Python

In order to facilitate the encoding of the warehouse game into Promela, a Python program was developed. This program takes the minimum information of the warehouse game:

- The dimensions of the grid, m and n
- The number of robots
- For each robot, a name, its starting position, load point and exit point.

From this, the Promela program is automatically generated, including all boilerplate and helper code, as well as the conversion of the LTL formulae into Promela syntax. The safety objective (1) is always used, and one of (2) or (3) can be chosen by the user depending on whether fairness is desired, or if only collective score should be considered.

5.2.1 Collective score

If we choose to optimise for collective score, then the following approach is used:

Algorithm 3 Score optimisation

$\gamma \leftarrow 1$ ▷ Set the initial total score target
$N \leftarrow 10$ ▷ The maximum number of times SPIN should be called
$final \leftarrow 0$ ▷ the highest score found
$i \leftarrow 0$
while $i \neq N$ **do**
  $total \leftarrow$ total mean-payoff from strategy
  if $total > \gamma$ **then**
    $\gamma \leftarrow total + 1$
  else ▷ no improvement from previous iteration
    $final \leftarrow total$
  end if
  $i \leftarrow i + 1$
end while

In practice, when iteratively increasing the score target, there are 2 challenges:

- If we find the highest possible score, trying to increase the score target beyond that is impossible, as discussed in section 4.3.2. This results in Promela exhaustively exploring the entire state space, which is exceedingly time-consuming.

- In the case of complex scenarios, higher scoring strategies may exist, but it may take an infeasible amount of time to synthesise them.

To combat these issues somewhat, the program allocates SPIN an amount of time to find a strategy at each iteration. If the time taken by SPIN exceeds this timeout, the search is abandoned and current best score (and corresponding strategy) are returned.

In some scenarios, the collective optimal may also be fair, because a strategy where all the agents are doing work may lead to the highest total as well. In simple scenarios, optimising the collective score enough leads to fairer strategies.

5.2.2 Fairness

If we instead wish for all agents to do some work, we can employ quantitative objective (3). In practice, finding fair strategies for arbitrary scenarios of the warehouse game is computationally very expensive. This makes iteratively increasing the score targets for each agent one-by-one very time consuming. In this implementation, only a basic target is set, without any iterative searching, namely the score target $\gamma_a = 1 \; \forall a \in Agt$. This guarantees a strategy profile where each agent scores at least once, but may possibly score more than once.

5.3 Visualiser

In order to view the paths of the agents, the Python program interprets the response of SPIN, and creates a visual colour-coded representation of the warehouse game. The use can then step through the simulation of the program on a step-by-step basis. One important function of the visualiser is that it detects when collisions occur, and this means that the value chosen for k is too small, i.e. the collision occurs after the number of time steps simulated in SPIN. This error means that the number of steps needs to be increased for the scenario being considered.

The agents each have a colour, and are represented by a circle of that color. Squares represent the load and exit points, which are also coloured to match their respective agents. Additionally, a tracker on the right tracks the

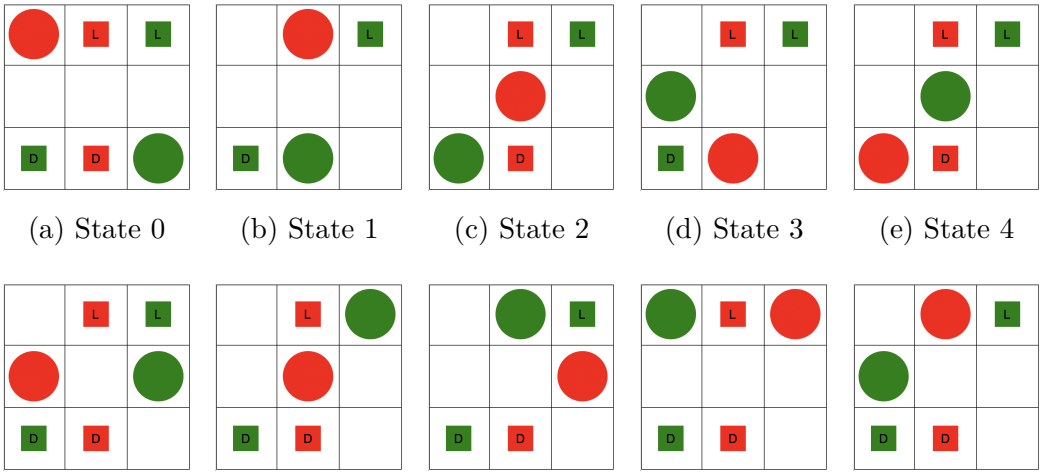
score of each agent, and whether or not the agent is currently carrying an item.

6 Experiments

In order to test the feasibility of the approach employed, various scenarios of the warehouse game were tested. In the testing, the number of agents as well as the size of the grid and positions of the load and exit points were altered. The results from some illustrative scenarios are provided here.

6.1 3x3 2 agents

This scenario involves a 3x3 grid with 2 agents. The load points and exit points are placed on opposite sides of the grid, meaning that the agents must go from one side of the grid to the other. For this example, the full path is shown below.



(a) State 0 (b) State 1 (c) State 2 (d) State 3 (e) State 4

(f) State 5 (g) State 6 (h) State 7 (i) State 8 (j) State 9

Figure 1: Full state space

Note that the state after state 9 (fig. 1j) is state 2 (fig. 1b), thus completing the loop. The overall path is then $\pi = s_0 s_1 (s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9)^\omega$.

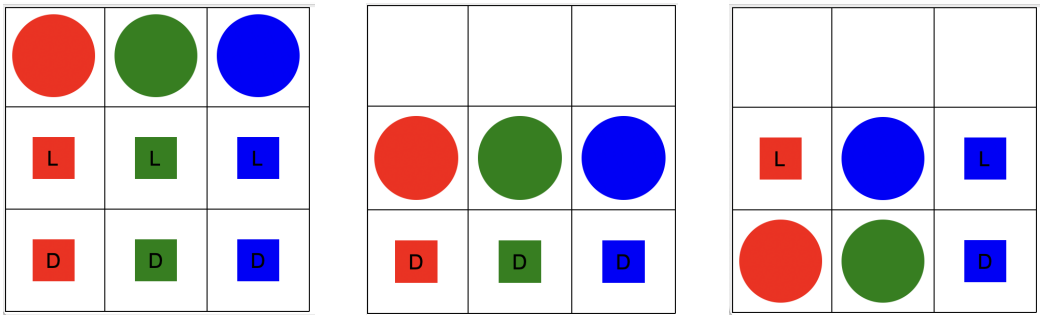
In this scenario, red has a mean-payoff of 7/50, and green has a mean-payoff of 6/50.

Though red has a shorter path, it is blocked from going up by green in state 3 (fig. 1d). Instead red goes to the left, and then continues going up,

and green takes a direct path to its load point (fig. 1e). This solution is found quickly, taking just 5 iterations.

6.2 3x3 3 agents

This scenario features a setup that would be very easy for a human to solve simply by inspection. Each agent only needs to go up and down in a straight line to make as many deliveries as possible. The approach employed here instead takes many iterations to arrive at the optimal solution. The first solutions generated only consider 1 or 2 agents, and do not find the shortest (straight) path.



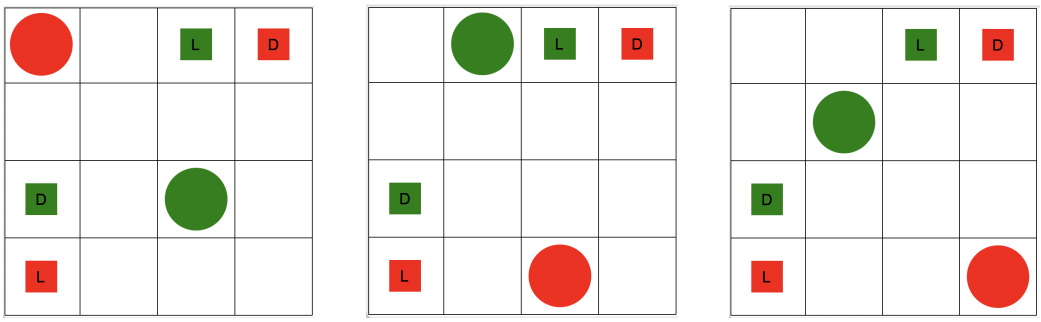
(a) Initial Position (b) Critical Position (c) After critical

Figure 2: Important states

Here even after 16 iterations, the optimal paths were found only for red and green. Blue detours to left, instead of making the delivery directly. This results in red and green having mean-payoffs of 24/50, and blue only 12/50.

6.3 4x4 2 agents

In this scenario, the extra space provided means that both agents are able to take the shortest path possible between the load point and the exit point. They never interfere with each other. The optimal solution is found in 6 iterations.



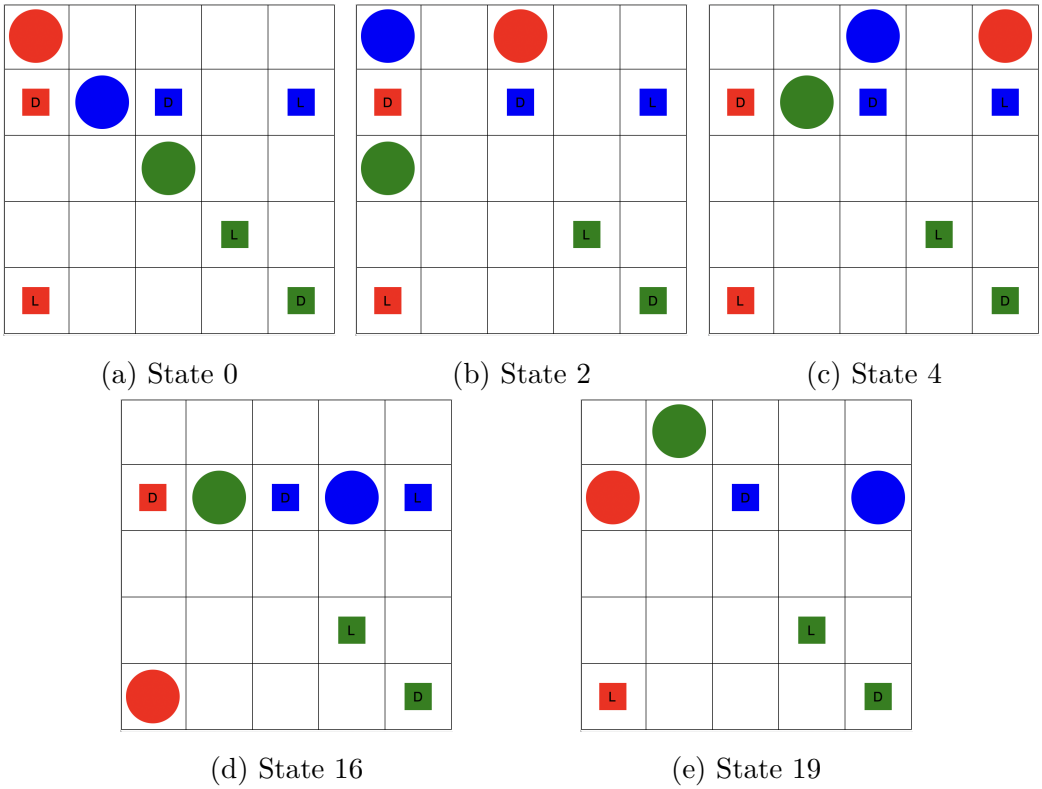
(a) Initial Position (b) Free space (c) Free space

Figure 3: Important states

The mean-payoff here is 4/50 for red, and 6/50 for green, since green has a shorter path.

6.4 5x5 3 agents

This scenario features a 5x5 grid with 3 agents. The agents have enough space to operate without hindering each other, but the approach employed is not capable of finding the optimal solution. Like in section 6.2, the synthesises solution is only able to generate scoring paths for 2 of the 3 agents, and in this case the paths are not optimal either.



(a) State 0 (b) State 2 (c) State 4

(d) State 16 (e) State 19

Figure 4: Important states

Even though red has a straight path downward to its load point, it takes a detour to the right (shown by figs. 4b and 4c), and eventually makes it to the load point in state 16 (fig. 4d). It scores soon after in state 19 (fig. 4e). Blue is able to score much earlier as it finds its load and exit points much faster. Green unfortunately gets stuck in a loop in the top-left corner, not scoring at all.

In this strategy profile, red achieves a mean-payoff of 3/50, green a mean-payoff of 0, and blue a mean-payoff of 8/50. This strategy profile is reached in 4 iterations, because the timeout is exceeded.

6.5 Limits

This approach nears its limitations when the grid is larger than 5x5 and 3 or more agents are involved. The required number of time steps k increases, and synthesising more optimal strategies becomes infeasible.

7 Conclusion

The warehouse game is a good example of multi-robot systems which are a common topic in the field of model checking. The warehouse game corresponds well to real life scenarios, particularly logistics like manufacturing, warehousing and deliveries.

We define the warehouse game and its components in a formal way, and use a novel approach which uses the well known model checking tool SPIN to synthesise strategy profiles. Strategy synthesis is not SPIN's intended purpose, and this becomes apparent in larger scenarios, but for smaller scenarios this approach works well because it possible to search the state space exhaustively.

Synthesising fair strategy profiles was very computationally expensive, so in most cases the alternative of optimising the collective score was used instead, however the approach does have the capability to synthesise fair strategy profiles.

We also provide an Python tool to facilitate the easy generation of scenarios and the associated Promela code, as well as a way to visualise the strategy profiles and paths generated by SPIN.

Finally we analyse some illustrative scenarios and provide analysis on the scenarios considered, which gives some idea as to the types of scenarios which this approach is able to handle, and some that it is not.

8 Future Work

This work could be extended in several ways. Keeping with the approach detailed, using a more efficient encoding of the warehouse game into Promela could make finding strategies faster, and make scenarios which are currently infeasible, feasible.

The approach used would also benefit from a more intelligent search. Currently, SPIN has to explore a large number of paths because move choices are made randomly at first. If we instead use some heuristic (such as manhattan distance) we could attempt to guide an agent towards its current goal

(either to load or drop off an item). This could vastly improve the results by reducing the search space, but in complex, tightly packed scenarios, this approach would not necessarily be better than the current approach, because the paths favoured by the heuristic may often be blocked.

Another approach would be to use a directed model checking approach like in [11]. The directed approach aims to find shorter counter-examples to the LTL formulae, which in our scenario would result in shorter (i.e. more efficient) paths.

Alternative operational models can also be considered. We could have a system where there is no space for the robots to move past each other on the warehouse floor. In this case we would need to modify the operational model to that 2 adjacent agents cannot swap places, because there is not enough space.

Finally, a different synthesis technique could be used. There are dedicated strategy synthesis tools, and outside of those, a modified search algorithm, or AI based approach could also be explored. The alternative approaches would then be compared against the approach employed here, both on a quantitative aspect (i.e. the scoring) and on how long strategy synthesis would take.

A Appendix

The Python program is freely available on Github.

References

[1] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi, "Reasoning about strategies: On the model-checking problem," *ACM Transactions on Computational Logic (TOCL)*, vol. 15, no. 4, pp. 1–47, 2014.

[2] J. Gutierrez, A. Murano, G. Perelli, S. Rubin, T. Steeples, and M. Wooldridge, "Equilibria for games with combined qualitative and quantitative objectives," *Acta Informatica*, vol. 58, no. 6, pp. 585–610, 2021.

[3] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28573–28593, 2018.

[4] O. Salzman and R. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems,"

in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1711–1715, 2020.

[5] N. Timm and J. Botha, "Model checking and strategy synthesis for multi-agent systems for resource allocation," in *Formal Methods: Foundations and Applications: 24th Brazilian Symposium, SBMF 2021, Virtual Event, December 6–10, 2021, Proceedings 24*, pp. 53–69, Springer, 2021.

[6] J. Pilecki, M. A. Bednarczyk, and W. Jamroga, "Synthesis and verification of uniform strategies for multi-agent systems," in *Computational Logic in Multi-Agent Systems: 15th International Workshop, CLIMA XV, Prague, Czech Republic, August 18-19, 2014. Proceedings 15*, pp. 166–182, Springer, 2014.

[7] A. Lomuscio, H. Qu, and F. Raimondi, "Mcmas: an open-source model checker for the verification of multi-agent systems," *International Journal on Software Tools for Technology Transfer*, vol. 19, pp. 9–30, 2017.

[8] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*, vol. 1003. Addison-wesley Reading, 2004.

[9] T. Stützle, H. Hoos, and A. Roli, "A review of the literature on local search algorithms for max-sat," *Rapport technique AIDA-01-02, Intellectics Group, Darmstadt University of Technology, Germany*, 2001.

[10] N. Timm, J. Botha, and S. Jordaan, "Max-sat-based synthesis of optimal and nash equilibrium strategies for multi-agent systems," *Science of Computer Programming*, vol. 228, p. 102946, 2023.

[11] S. Edelkamp, A. L. Lafuente, and S. Leue, "Directed explicit model checking with hsf-spin," in *International SPIN Workshop on Model Checking of Software*, pp. 57–79, Springer, 2001.