Bachelor Project

Czech
Technical
University
in Prague

F3 Faculty of Electrical Engineering
Department of Cybernetics

Multi-Agent Path Finding with Human Presence

Sofie Konvalinová

Supervisor: Ing. David Zahrádka
Field of study: Cybernetics and Robotics
November 2025

Acknowledgements

I would like to thank my supervisor, Ing.David Zahrádka, for leading consultations, valuable feedbacks, and had a lot of patience with me.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography. Prague, November , 2025 Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu. V Praze, . listopadu 2025

Abstract

As robots become common in shared workspaces, we need better safety measures. Standard planning algorithms avoid crashes but can accidentally surround a human worker, blocking their escape route. This thesis solves this problem by introducing the Safety-Aware Large Neighborhood Search (LNS) algorithm.

Our method runs a strict check at every time step to ensure the human always has a clear path to a Safety Zone. If the system detects a potential trap, it immediately forces the robots to move out of the way. Experiments show that this approach guarantees safety with minimal impact on efficiency. Even in crowded scenarios with 100 agents, the solution cost increased by only 0.3% to 1.5%.

Keywords: Multi-Agent Path Finding

Supervisor: Ing. David Zahrádka

Abstrakt

S rostoucím počtem robotů ve sdílených prostorech potřebujeme lepší bezpečnostní opatření. Standardní plánovací algoritmy sice zabrání srážkám, ale mohou člověka náhodně obklíčit a zablokovat mu únikovou cestu. Tato práce řeší tento problém pomocí nového algoritmu Safety-Aware Large Neighborhood Search (LNS).

Moje metoda v každém kroku simulace ověřuje, zda má člověk volnou cestu do bezpečné zóny. Pokud systém detekuje, že by mohl být člověk uvězněn, okamžitě přinutí roboty uhnout. Experimenty ukazují, že tento přístup garantuje bezpečnost s minimálním dopadem na efektivitu. I v přeplněném prostředí se 100 agenty vzrostla cena řešení pouze o 0,3 % až 1,5 %.

Klíčová slova: Multiagentní plánování cest

Překlad názvu: Multiagentní plánování cest za přítomnosti člověka

Contents

Figures

Tables

Chapter 1

Introduction

Consider a group of autonomous agents moving in a shared environment like a warehouse. Every agent has a specific task: to travel from a start location to a target destination. However, as we add more agents to the fleet, the risk of accidents gets much higher because there is less free space. The challenge of planning these paths safely, so that no agents collide, is known as Multi-Agent Path Finding (MAPF). As the number of agents grows, coordinating their movements becomes increasingly difficult. Therefore, we need special algorithms to make sure everyone gets to their destination safely and without long delays.

While current studies offer a wide array of solutions focusing on efficiency, optimality, and robustness [1, 2, 3] of robot-to-robot coordination, less attention has been paid to the interaction between autonomous fleets and human workers sharing the same workspace [4]. Some recent studies have started to address the uncertainty of human behavior[4], attempting to predict where a person might move next using probabilistic models. However, while these methods are efficient, they do not strictly enforce the safety boundaries needed to protect human workers in unpredictable situations.

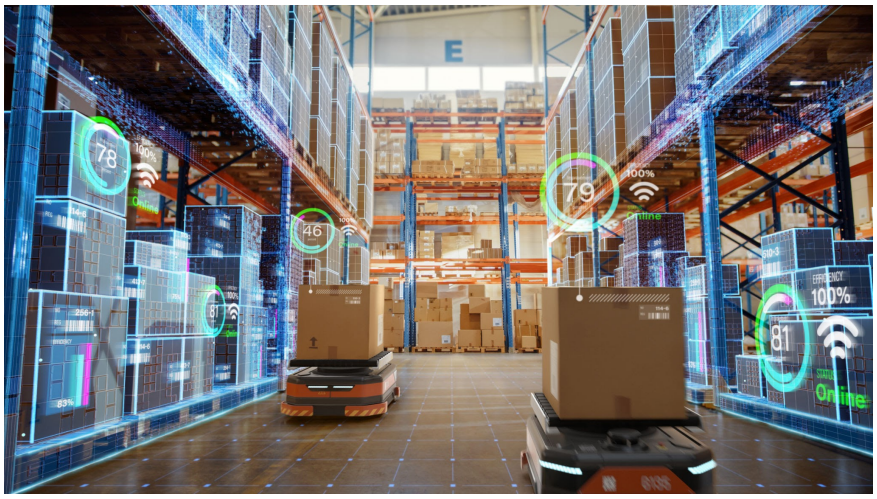

Figure 1.1: A visualization of a modern warehouse with fleet of autonomous robots. Source: https://www.supplychainbrain.com

This thesis focuses on generating safe paths for robots operating alongside human workers. Specifically, it aims to solve the Multi-Agent Path Finding problem in a way that guarantees the safety of moving technicians within the workspace. An example warehouse can be seen in Fig. 1.1. The primary goal is to extend current planners to strictly guarantee that the human worker always has a clear trajectory to the safety zone.

1.1 Contributions

The main contributions of this thesis are as follows:

- **Formulation of a Human-Safety MAPF:** The thesis introduces a new version of the MAPF problem that includes a human worker. Unlike standard methods, we add a strict rule ensuring that a safe path to the exit is always available for the human.
- **Implementation:** I developed a simulation scenario representing a mixed warehouse environment. To calculate the paths for the robots, we use an existing method [5]. My work focuses on the interaction with a human maintenance worker and the challenges of dynamic obstacle avoidance.
- **Testing on Benchmarks:** We tested the algorithm using standard benchmark instances. The goal was to compare our method with existing solvers and analyze the impact of the escape route mechanism.

Chapter 2

Related Work

This chapter outlines existing research in optimal MAPF solvers, dynamic environments, and human-robot safety standards. The analysis highlights the lack of guaranteed escape mechanisms and defines the research gap which this project intends to fill.

2.1 Optimal MAPF Solvers

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths for a set of agents from their start locations to their goal locations. The problem is widely applicable in automated warehousing, traffic management, and computer games [6].

2.1.1 Conflict-Based Search (CBS)

Older methods tried to plan paths for all robots at the exact same time using the standard A* algorithm. The problem is that with many robots, the math becomes too difficult for computers to handle quickly [7].

To fix this, **Conflict-Based Search (CBS)** [2] was created. The idea is simple: instead of coordinating everyone perfectly from the beginning, let each robot find its own best path first, as if it were alone. Then, the system checks if any robots crash. It only fixes the specific spots where paths cross. This saves a lot of computing power because the system doesn't have to calculate every move for every robot at once.

CBS operates on two levels:

- **High-Level:** Acts like a manager. It looks for collisions between robots. If two robots crash, it creates a new rule (constraint) to stop that specific crash from happening again.
- **Low-Level:** Acts like a worker. It finds a path for a single robot that follows the rules given by the High-Level manager.

Since finding the perfectly optimal solution can still be slow, variants like **Enhanced CBS (ECBS)** [8] are often used. ECBS is a **bounded-suboptimal** algorithm, which means it finds a solution much faster by

allowing the path to be slightly longer than the best one, but within a specific limit.

2.1.2 Prioritized Planning

A common alternative is **Prioritized Planning (PP)** [1]. Where each robot is given a number based on importance. Then, PP finds a path for the agent (robot) with the highest priority. This agent becomes a dynamic obstacle, and the algorithm/method continues with the next robot.

However, it has a major flaw: it is not guaranteed to find a solution [9]. Sometimes, a lower-priority robot gets blocked by the robots before it and cannot find a path, even if a solution actually exists. In safety-critical situations, this is risky because the system might fail to calculate a path when it is needed most.

2.2 Dynamic Environments and Replanning

Standard MAPF usually assumes a static world where we know where every obstacle is before we start. But real warehouses are dynamic—machines move around, and people walk through the aisles.

2.2.1 Replanning Approaches

The most common way to handle surprise obstacles is Replanning. Algorithms like **Lifelong MAPF** calculate new paths every few seconds or whenever a robot gets stuck [10].

The robot only changes its plan after it sees an obstacle. For human safety, this is not good enough. By the time the robot realizes it needs to move, the robots might have already surrounded the human, blocking their way out.

2.2.2 Robustness and k-Delay

Another method is to make plans that can handle delays. This is called **k-robustness** [3]. It ensures robots don't crash even if they are a few seconds late.

While this helps with small mistakes or delays, it doesn't guarantee that a path to the exit will stay open for a person who is moving unpredictably. It makes the plan stronger against delays, not against a moving human.

2.3 Human-Robot Safety in Shared Workspaces

Ensuring human safety in robotized environments is a **fundamental requirement** for modern industry.

2.3.1 Local Collision Avoidance

Most safety systems today use sensors like cameras or lasers (LiDAR). Robots use these to see nearby objects. If a human gets too close, the robot stops or moves aside.

Most safety systems today use sensors like cameras or lasers (LiDAR). Robots use these to see nearby objects. If a human gets too close, the robot stops or moves aside using reactive methods like ORCA (Optimal Reciprocal Collision Avoidance) [11].

This logic prevents a physical crash, but it doesn't solve the problem of getting trapped. If many robots stop around a human to avoid hitting them, they can accidentally create a wall that blocks the human from leaving.

2.3.2 Predictive Human Modeling

Advanced research tries to guess where a human will walk using Artificial Intelligence (Machine Learning) [4]. If robots know where the human is going, they can stay out of the way.

However, in an emergency, people act unpredictably and might run in random directions. These computer guesses are not 100% certain, so they cannot guarantee the absolute safety needed to save a life.

2.4 Gap in Research

Based on the literature review, we identified a significant gap. Current research is excellent at coordinating robots to work efficiently and avoiding immediate crashes. However, there is no global strategy that treats **human evacuation** as a strict, mandatory rule.

Existing solvers focus on avoiding collisions, not on keeping a path open to an exit. Because of this, standard algorithms might create efficient plans that accidentally trap the human agent in a **deadlock**. This thesis fixes this problem by building a guaranteed escape route directly into the planning logic, ensuring that safety is just as important as efficiency.

Chapter 3

Problem Formulation

In this chapter, we define exactly what problem we are solving. We set the map, the inputs, the players (robots and humans), how time works, and what counts as a valid solution.

3.1 The Environment

To control the agents, we first discretize the continuous warehouse environment into a grid. Formally, we represent the workspace as a graph $G = (V, E)$.

The **Vertices (V)** represent the individual cells of the grid where an agent can stand. The **Edges (E)** represent the connections between these tiles. If two tiles are connected, a robot can move between them.

However, not all tiles are accessible. We define a set of **Static Obstacles (X)**, such as walls or shelving units, visualized as **black tiles**. Robots are strictly forbidden from entering these vertices. All other vertices are considered **Free Space** (V_{free}).

A crucial part of our environment is the **Safety Zone** (Z). This is a specific set of vertices representing emergency exits or safe areas. The primary safety goal of our system is to ensure that the human worker can always reach one of these vertices.

To better visualize these definitions, Figure 3.1 displays a representative instance of the problem. This specific environment is populated by a fleet of **20 autonomous agents**. To visualize their individual trajectories, each agent is assigned a **unique color**, with lines indicating their planned paths from their specific start positions to their **final target goals**. The **Human**, showed by the **cyan icon** (top-left), shares the workspace with the robotic fleet. Finally, the designated **Safety Zone** is clearly marked as a **light brown door** (at right side). This setup serves as the primary testing ground for our safety verification algorithms.

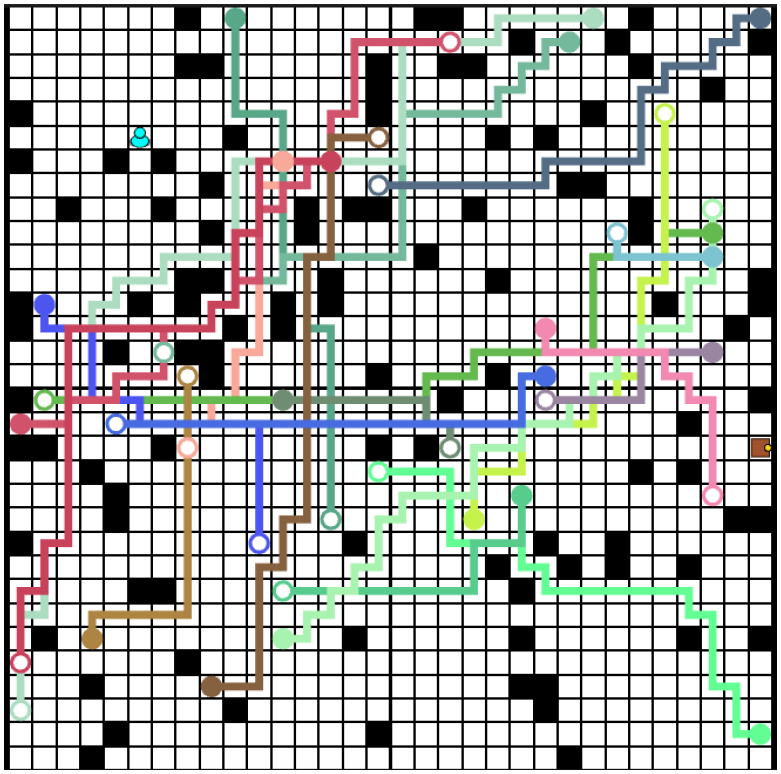

Figure 3.1: Visualization of the grid environment.

3.2 Time and Movement

In the real world, time flows continuously. However, to make planning easier, we divide time into distinct logical steps. This approach makes the calculations much simpler but remains accurate enough for warehouse operations.

Time moves forward in steps (ticks), marked as $t = 0, 1, 2, \ldots$. The system is synchronized, so at every step, all agents (robots and the human) move at the exact same time. We assume that moving to a neighboring tile takes exactly one unit of time. At any given step, an agent at position v has two possible choices:

- **Move:** The agent transitions to an adjacent vertex v' if an edge (v, v') exists.
- **Wait:** The agent remains at the current vertex v.

Example of Discrete Movement

To better understand this concept, Table 3.1 shows a real trajectory of a single robot (Agent 0) taken from our experimental data. At each time step t, the agent is at a specific grid **coordinate** (x, y) and performs exactly one action to reach the next state.

Time (t)	Position (x, y)	Next Position	Action Performed
5	$(16, 16)$	$(16, 17)$	**Move** (Down)
6	$(16, 17)$	$(16, 17)$	**Wait**
7	$(16, 17)$	$(17, 17)$	**Move** (Right)
8	$(17, 17)$	$(17, 18)$	**Move** (Down)

Table 3.1: A step-by-step example of Agent 3's movement, demonstrating a **Wait** action.

This illustrates that a continuous path is actually a sequence of discrete checkpoints.

3.3 Agent Behavior

The environment contains two different types of agents. They behave in completely different ways.

- **The Autonomous Robot (a_i):** We define a single robot as a **cooperative agent**. Its only job is to follow a planned path from a Start location (s_i) to a Goal location (g_i). The robot is fully deterministic. This means it follows the computer's instructions exactly and never changes the path on its own.
- **The Robotic Fleet (A):** The system controls a team of k robots, written as $A = \{a_1, a_2, \ldots, a_k\}$. Each robot has its own goal, but the whole team must work together. We synchronize their movements to make sure they do not crash into each other.
- **The Human Agent (H):** Unlike robots, the human is an **uncontrolled element**. We assume their movement is unpredictable ("non-deterministic"), as illustrated in Figure 3.2. They do not follow our orders. They might decide to move to any neighboring square or stop suddenly at any time.

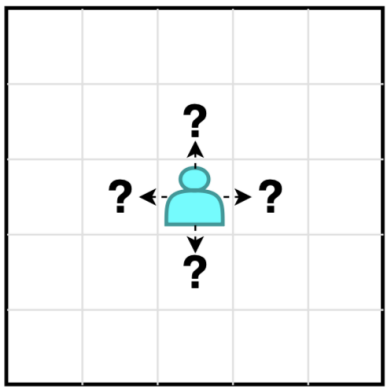

Figure 3.2: Visualization of the Human Agent's non-deterministic behavior.

Chapter 4

Method

This chapter describes the algorithmic framework used to solve the MAPF problem with a specific focus on human safety. We present a solution that adds a strict safety check to the standard pathfinding process. The core idea is simple: at every single time step, we check if the human can escape. The architectural blueprint of our solution is presented in **Section 4.1**, which provides a higher level overview of the entire system. It explains the iterative loop of the solver and how it manages the interaction between the initial pathfinding and the subsequent safety checks. Following the system overview, **Section 4.2** details the mathematical core of the thesis. It defines the concept of Reachability Analysis, which is strictly used to evaluate whether a specific state is safe. This section demonstrates how we prove that a valid escape path exists for the human at every time step of the simulation. Finally, to ensure the solution is not only valid but also of high quality, we adopt the codebase provided by Jan Chleboun [5] for the Large Neighborhood Search (LNS) algorithm. This iterative meta-heuristic does not recalculate the entire problem from scratch. Instead, it selectively identifies and repairs specific agent trajectories that are either unsafe or inefficient. The specific mechanics of the destroy and repair operators are discussed in detail in **Section 4.3**.

4.1 Algorithmic Framework

The proposed system operates as an iterative solver. Its primary goal is to balance two competing objectives: the efficiency of the robotic fleet and the absolute safety of the human worker.

Unlike traditional MAPF solvers that only focus on preventing robot-to-robot collisions, our framework introduces a dedicated "Safety Supervisor" layer. This layer acts as a strict filter—it rejects any plan that could potentially trap the human agent, even if that plan is highly efficient for the robots.

The overall process is divided into three logical steps, which form a continuous loop:

1. **Initialization (The First Draft):** First, the system needs a starting point. We generate a initial plan for all robots using a fast method called

Prioritized Planning. In this phase, the robots simply try to reach their goals without crashing into each other. However, this initial plan is not yet verified for human safety. It serves only as a baseline solution that we will improve in later steps.

2. **Safety Validation (The Check):** Once a plan exists, the system performs a rigorous safety check. It iterates through every time step of the generated plan (from the beginning to the end). At each step, it performs a *Reachability Analysis* to answer a simple question: "Does the human have an open path to the Safety Zone?" If the answer is "No" at any point, the system identifies exactly which robots are blocking the way and marks the plan as unsafe.

3. **Iterative Optimization (The Fix):** Finally, we use the Large Neighborhood Search (LNS) algorithm to fix the problems found in the previous step. LNS works on a "Destroy and Repair" principle. It selects the specific robots that are causing safety violations (or moving inefficiently) and deletes their current paths. Then, it attempts to find new, better paths for them. This cycle repeats until a solution is found that is both fullly safe for the human and efficient for the robots.

4.2 Safety Verification Mechanism

To keep the human worker safe, we use a method called **Reachability Analysis**. Standard collision avoidance only stops robots from hitting people. It does not prevent robots from surrounding a person. Our method ensures the human is never trapped. The main rule is simple: at any moment, there must be at least one clear path to the Safety Zone.

The check at each time step t works in three steps:

1. **Freeze the Time:** We view the simulation as a series of static pictures. At time t, we assume all robots are frozen at their positions (x, y).

2. **Identify Walls:** These frozen robots act like temporary walls. They create a temporary maze. The human must be able to walk through it.

3. **Check the Path:** We run the SIPP (Safe Interval Path Planning) algorithm starting from the human's position H_t. SIPP is an advanced version of A*. It is highly efficient for dynamic environments. Since we use SIPP for the robots, we use the same logic to quickly check the human's escape route.

If the SIPP solver finds no path, we mark the state as **Unsafe**. This means the robots have blocked the human. They created a trap.

Algorithm 1 Safety-Aware LNS Algorithm

Require: Map G, Agents A, Human Path P_H, Safety Zone Z
Ensure: Safe Plan S

1: $S \leftarrow$ InitialSolution(A)
2: **while** $iter < MaxIterations$ **do**
3: ▷ 1. Standard Optimization
4: $S_{new} \leftarrow$ LNS_Step(S) ▷ Destroy and repair random agents
5: ▷ 2. Safety Check
6: $is\_safe \leftarrow$ **true**
7: **for** $t \leftarrow 0$ **to** $Makespan(S_{new})$ **do**
8: $R_t \leftarrow$ GetRobotPositions(S_{new}, t)
9: **if** ¬SIPP_FindPath($from = P_H[t], to = Z, obstacles = R_t$) **then**
10: $is\_safe \leftarrow$ **false**
11: $blocking\_robot \leftarrow$ FindBlocker($R_t, P_H[t]$)
12: **break**
13: **end if**
14: **end for**
15: ▷ 3. Fix or Accept
16: **if** is_safe **then**
17: **if** Cost(S_{new}) < Cost(S) **then**
18: $S \leftarrow S_{new}$ ▷ Better safe solution found
19: **end if**
20: **else**
21: ▷ Safety Violation: Fix only the blocking robot
22: DestroyPath($S_{new}, blocking\_robot$)
23: Replan($S_{new}, blocking\_robot$, constraint: avoid blocking)
24: $S \leftarrow S_{new}$
25: **end if**
26: $iter \leftarrow iter + 1$
27: **end while**
28: **return** S

Algorithmic Logic

Algorithm 1 shows our safety checks. We add them to the Large Neighborhood Search (LNS). The process has three parts.

In **Phase 1** (lines 5–7), the solver does a normal LNS step. It creates a new candidate plan S_{cand}. It does this by changing the paths of a few random agents. This makes the global solution more efficient.

In **Phase 2** (lines 9–17), we check if the new plan is safe. We check every time step t. The function FindPathToZone (line 13) uses the **SIPP algorithm** to find a way out. If the human cannot reach the exit, we stop checking.

In **Phase 3** (lines 18–22), we make the final decision. If the candidate plan S_{cand} passed all safety checks ($isSafe$ = true), we accept it and update

the current solution. However, if the plan failed even a single check, we immediately reject S_{cand} and revert to the previous valid solution. This ensures that the system never switches to an unsafe state.

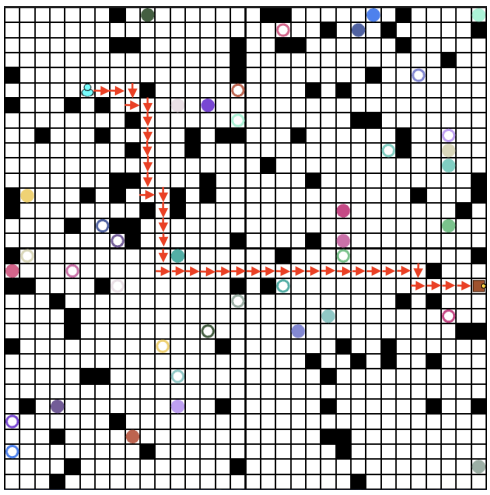

Figure 4.1: Planned path of human to the doors.

On Figure 4.1 is an example of a valid evacuation route. The system verifies that a path exists from the human's current position to the designated Safety Zone, among static obstacles.

4.3 Optimization via Large Neighborhood Search (LNS)

Finding the optimal solution for many robots in a crowded warehouse is a very difficult computational problem. A full recalculation of the joint path plan is too time-consuming for real-time operations. Therefore, we adopt the **Large Neighborhood Search (LNS)** framework based on the codebase provided by Jan Chleboun [5].

LNS is an iterative method. This means it improves the solution step by step. It does not try to fix the whole plan at once. Instead, it continuously fixes small parts of the plan that are inefficient.

The algorithm works in a loop consisting of two main operations:

1. **Destroy:** The system selects a small group of agents and deletes their paths.
2. **Repair:** The system recalculates new paths for these agents using the SIPP solver.

4.3.1 Adaptive Destroy Strategy

In our implementation, we do not simply choose random agents to destroy. The solver uses an **Adaptive** mechanism. It chooses from different strategies

based on how well they worked in the past:

- **Random Destroy:** Selects agents completely randomly.
- **Intersection Destroy:** Targets agents that are crowded in intersections, where traffic jams often happen.
- **Random Walk:** Selects agents that are close to each other.

If a strategy successfully improves the solution, the algorithm gives it a higher "weight" and uses it more often.

4.3.2 Standard LNS Algorithm

The structure of the LNS solver is formalized in Algorithm 2. It starts with a quick initial solution using *Prioritized Planning* and then continuously attempts to lower the total cost.

Algorithm 2 The Large Neighborhood Search Algorithm

Require: Initial Map G, Agents A, Time Limit t_{limit}
Ensure: Improved solution S_{bsf}

$S_{init} \leftarrow$ PrioritizedPlanning(A) ▷ Generate initial solution
$S_{bsf} \leftarrow S_{init}$
$i \leftarrow 0$
while not CheckTermination(i, t_{limit}) **do**
  $S_{new} \leftarrow S_{bsf}$ ▷ Copy current best solution
  ▷ 1. Destroy Phase (Adaptive)
  $S_{new} \leftarrow$ Destroy(S_{new})
  ▷ 2. Repair Phase (SIPP)
  $S_{new} \leftarrow$ Repair(S_{new})
  ▷ 3. Compare Cost
  if CalculateCost$(S_{new}) <$ CalculateCost(S_{bsf}) **then**
    $S_{bsf} \leftarrow S_{new}$ ▷ Accept improvement
    UpdateWeights(success)
  else
    UpdateWeights(fail)
  end if
  $i \leftarrow i + 1$
end while
return S_{bsf}

Chapter 5

Experiment results

To validate the proposed Safety-Aware LNS algorithm, we created a series of experiments. The implementation was written in C++ and executed on **Apple M4 processor with 24 GB of unified memory**. We utilized the map shown in Figure 3.1, simulating a standard warehouse environment.

5.1 Experiment 1: Impact of Agent Density and Map Topology

The goal of this experiment was to test the Safety-Aware LNS algorithm. We checked if it could find a safe solution for the human in different environments and with different amounts of traffic.

All tests were conducted on grid maps with dimensions of 32×32 cells. To simulate different conditions, we selected three different map topologies:

- **Empty:** An open space without obstacles, allowing maximum freedom of movement for agents.
- **Random:** A map with randomly placed obstacles.
- **Room:** A structured map with a narrow corridors. This topology simulates a small warehouse environment.

Agents (k)	Empty	Random	Room
10	✓	✓	✗
50	✓	✗	✗
100	✗	✗	✗

Table 5.1: Impact of density and map topoogy.

For each map type, we tested scenarios with $k = 10, 50$, and 100 agents.

The results of this test are summarized in Table 5.2. The symbol ✓ indicates that a safe path for the human was successfully found. While ✗ signifies that the solver could not find a valid solution because of the time limit or that the human was blocked by agents without a valid escape route.

- On the **Empty** map, the algorithm successfully handled up to 50 agents. The failure at 100 agents means that even in an open space, extreme agent density creates a high-density traffic. That prevents the human from reaching the goal safely.
- The **Random** map shows a lower tolerance for traffic. While 10 agents were succesfull, increasing the count to 50 led to failure. With less free space available due to obstacles, robots struggle to effectively give way to the human.
- The **Room** map proved to be the most challenging scenario. The solver failed to find a safe solution even for the lowest density of 10 agents. This result highlights the difficulty of environments (simulated warehouse), where narrow doors can be easily blocked by a single agent, making the human's path unsafe almost immediately.

5.2 Experiment 2: The "Cost" of Safety

In this experiment, we measure how safety rules affect the efficiency of the whole system. We use a metric called *Cost Increase*. This is the percentage difference between the **Safety-aware** solution and the **Baseline** solution (where agents ignore the human).

Agents (k)	Baseline SOC	Safety-Aware SOC	Cost Increase (%)
10	305	306	+0.33%
70	2213	2220	+0.32%
100	3608	3662	+1.50%

Table 5.2: Comparison of solution costs.

$$\text{Cost Increase } (\%) = \frac{SOC_{safety} - SOC_{baseline}}{SOC_{baseline}} \times 100 \tag{5.1}$$

The experiment was conducted on the **Room (32x32)** map, which features narrow streets forcing agents to interact with the human. We tested agent densities of $k = 10$, 70, and 100. The results are presented in Table 5.2.

Discussion of Results

The results shows that the algorithm works efficiently, but the percentage values are uninformative. The percentage was calculated using Equation 5.1).

Figure 5.1 illustrates the performance of standard solvers. For low and medium traffic ($k = 10$ to 70), the operational cost remains very low (overhead around 0.3%). However, this efficiency is misleading. The low cost indicates that the agents are **not actively deviating** to preserve a safety corridor for the human. Instead, they prioritize their own path optimality, which creates

a high risk of trapping the human worker, as the system does not account for the necessary escape routes.

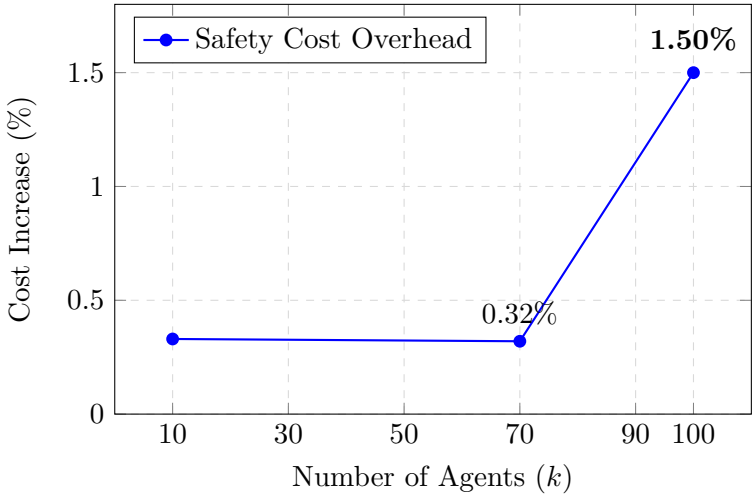

Figure 5.1: The relation between agent density and the safety cost overhead on the Room map.

Chapter 6

Conclusion

In this thesis, we addressed the critical challenge of ensuring human safety in shared robotic workspaces. As automated fleets become larger and more autonomous, standard collision-avoidance systems are not able to guarantee that a human worker will not be trapped by the robots. These standard systems typically focus only on efficiency and preventing physical crashes. They do not realize that by surrounding a person, they might cut off their only way out. To solve this, we proposed the **Human-Safety Large Neighborhood Search (LNS)** algorithm.

The core contribution of this work is the integration of a strict **Reachability Analysis** into the pathfinding process. This acts as a continuous safety check running in the background. Unlike standard methods, our system actively verifies at every single moment if the human can reach the Safety Zone. If no path exists, the system does not just fail silently. Instead, it precisely detects the specific time step where the problem occurred and identifies the specific robot causing the blockage.

Future Work

While the current implementation do not solves the defined problem, there is room for improvement.

Future work should focus on proactive planning. The autonomous fleet should calculate the human's escape route directly during the initial planning phase. This ensures that the exit path is never blocked at any time step

Bibliography

[1] D. Silver, "Cooperative pathfinding." in *AIIDE*, 2005, pp. 117–122.

[2] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI'12. Toronto, Ontario, Canada: AAAI Press, Jul. 2012, pp. 563–569.

[3] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Havens, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding," *Applied Intelligence*, vol. 50, pp. 889–909, 2020.

[4] P. A. Lasota and J. A. Shah, "A safe and efficient motion planner for robot manipulation in human-robot collaboration," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 299–306, 2017.

[5] J. Chleboun, "Large neighborhood search for multi-agent path finding," Master's Thesis, Czech Technical University in Prague, 2025.

[6] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent path finding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, 2019.

[7] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[8] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[9] M. Čáp, P. Novák, A. Kleiner, and M. Selecký, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[10] H. Ma, J. Li, T. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proceedings of the 16th*

Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 837–845.

[11] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.