# AUSTRIAN MARSHALL PLAN FOUNDATION

## Final Report

Salzburg University of Applied Sciences
Carnegie Mellon University

**Cornelia Ferner**

| | |
|---|---|
| Supervisor FHS: | DI Dr. Robert Merz |
| Supervisor CMU: | Prof. Howie Choset, Ph.D. |

Salzburg, October 2012

## Abstract

The complexity of multirobot systems increases exponentially with every additional robot. While single robot path planning problems are effectively solved using the A* algorithm [11], multirobot planners usually are either complete and optimal but computationally infeasible (coupled algorithms), or require less computation time but are not guaranteed to find a solution (decoupled algorithms).

In this thesis, a new multirobot path planning algorithm called *OD-M** is proposed. It combines the strengths of two existing planners: Subdimensional Expansion [29] and Operator Decomposition [27]. The new algorithm OD-M* constructs a search space of variable dimensionality, tailored to the given problem. When planning the paths within the low-dimensional space, promising low-cost nodes are favored. Additionally, the new optimal algorithm is embedded into a path planning framework called Independence Detection [27], a high-level planner to couple robots in collision, resulting in the *ID with OD-M** algorithm.

## Preface

Looking back at the past year makes me aware of having experienced something extraordinary and of great value: Dr. Merz, professor of mechatronics at the University of Applied Sciences in Salzburg (FHS) within the computer science department, and Mr. Enner, a formal Marshall Plan scholar of the FHS currently affiliated with the Carnegie Mellon University (CMU), initiated my stay at the CMU. After gathering information about my host university, I soon realized the great opportunity that the Marshall Plan Foundation (MPF) offers: I would be able to conduct research at one of the most renowned universities in the world, probably THE university when it comes to robotics. The financial support would guarantee a continuous stay of 5 months, enabling me to write my master thesis abroad.

Working with Howie Choset, one of the leading researchers in the field of mobile robotics, at his Biorobotics Lab within Carnegie Mellon's School of Computer Science gave me enriching insight into the current state-of-the-art research not only in mobile robotics, but also in medical robotics and - on the theoretical side - in path planning for complex robot systems. I took responsibility in two projects and specialized on path planning in multirobot systems - the subject of my master thesis. We developed an algorithm that solves path planning problems for complex systems of up to 40 robots in feasible runtime while producing cost-optimal paths.

Based on the work conducted for my thesis, we were also able to write and submit a conference paper for ICRA 2013 (International Conference on Robotics and Automation) that pursues and extends the ideas presented in my master thesis. After acceptance of the paper, we will have the opportunity to present our findings at the conference held in Karlsruhe in May 2013.

Currently, I am working at the University of Applied Sciences Salzburg within the mechatronics department where I have the chance to deepen my knowledge in the field of robotics. My research is now focused on human-machine interactions where we try to define new ways of "communicating" with robots.

Finally, I want to express my gratitude again to the responsible persons of all of the three institutions (FHS, CMU and MPS) who enabled my stay at CMU. The following report gives insight in my research at the Biorobotics Lab and is contains extracts of my thesis.

# Contents

# 1 Introduction

Robots are often used to carry out tasks humans do not want to or cannot do. Examples include dangerous tasks such as mining or search and rescue, high-precision tasks or tasks in inaccessible environments. Although a single robot can successfully execute the jobs mentioned above, even more benefits can be achieved in multirobot systems.

Imagine a human chain fighting a fire: While a single person carrying buckets of water would have no chance in controlling the fire, many people passing on the buckets are much more effective and will eventually stand a chance.

Similarly, multiple robots have advantages over a single robot: In multirobot systems, multiple tasks can be accomplished simultaneously. Multiple robots increase the fault tolerance and robustness of a system. Moreover, sensor coverage can be guaranteed. Typical fields of application for multiple robots are industrial assembly, search and rescue, surveillance or military tasks such as bomb demining. To be more specific, tasks for multirobot systems may include the assembly of supporting structures in collapsed buildings, the removal of contaminants or the search for disaster victims.

## 1.1 Motivation

In systems including more than one robot, it is necessary to coordinate the individual robots such that they reach their goal positions without running into collisions with obstacles or other robots. The single robot path planning problem is usually solved using the search algorithm A* [11]. Planning for multiple robots becomes more complex, leading to an exponential growth in the size of the configuration space with every additional robot. Multirobot path planning algorithms thus are not guaranteed to find a solution in feasible computation time. The challenge lies in computing a solution within an adequate runtime on common computers where all found paths have lowest costs possible.

A multirobot path planning algorithm has to ensure low runtime to prevent downtime of the system. Moreover, the algorithm should be executable on a common computer without special requirements. Additionally, the low-cost solution should prevent long detours and decrease the execution time. Algorithms for multirobot path planning usually either guarantee to find an optimal path (coupled algorithms) which is only applicable for systems with a small number of robots, or reduce the complexity of the problem (decoupled algorithms) where the algorithms are not guaranteed to find a solution.

Thus, the goal is to provide an optimal multirobot path planning algorithm with feasible computational runtime.

Recently, Wagner presented M* [29], a powerful algorithm that overcomes the disadvantages of both coupled and decoupled algorithms. However, the intention is to further increase the performance of M* regarding both the success rate and the runtime.

## 1.2 Contribution

Combining M* with Operator Decomposition (OD) [27], a variant of A* for multirobot path planning, results in the algorithm *OD-M**. OD-M* is additionally coupled with the Independence Detection (ID) framework [27], a high-level multirobot planning approach that groups colliding robots which results in the new algorithm *ID with OD-M**. ID with OD-M* is an approach that guarantees to provide optimal solutions in feasible runtime and outperforms existing multirobot planning algorithms such as basic OD.

For the simulations presented in the thesis, a variant of M*, namely recursive M* (rM*), which offers a different way of handling robots in collision, is used. This results in the algorithms OD-rM* and ID with OD-rM*, respectively.

## 1.3 Outline

In the following chapters, at first single robot path planning will be discussed in more detail: The concept of a configuration space is explained and basic graph search techniques are outlined. A detailed description of A* follows. Next, the curse of dimensionality - a phenomenon in high-dimensional spaces - will be discussed to understand the problem of planning in multirobot systems.

The subsequent chapter focuses on existing multirobot path planning algorithms and their characteristics and differences. Based on these approaches, the new algorithm OD-M* is explained and simulation results are provided. In the conclusion, also future developments and the next reasearch steps are outlined.

Parts of this thesis, especially the chapters concerning multirobot path planning algorithms, are based on [10] but with additional information about the single robot path planning problem and a detailed discussion of the algorithm ID with OD-M*.

# 2 Path Planning for a Single Robot

Path planning is the task to find a path from a robot's start position to its goal. Multirobot path planning is a generalization of single robot path planning, as multiple robots can be treated as one composite robot. Thus, any general algorithm for solving the single robot path planning problem can be applied to the multirobot problem, but would eventually lead to computational infeasibility. However, most multirobot planning techniques depend to some extent on single robot path planning algorithms.

As the path planning problem becomes more complex the more robots are involved, this chapter aims to introduce the fundamental concepts of path planning in single robot systems. A* is one of the most important algorithms for solving the single robot planning problem on a graph. Other algorithms ultimately rely on A*. Besides A*, there are many other planners that use a graph as underlying technique.

## 2.1 Graph Search

For computing collision-free paths for a robot in its environment, the configuration space can be represented as a graph. A graph $G$ is a collection of nodes $N$ and edges $E$; $G = \{N, E\}$. Nodes (or vertices) correspond to locations that the robots can visit, whereas an edge (or an arc) represents a connection between two nodes. If a robot can traverse from node $N_1$ to node $N_2$ and vice versa, we call the graph *undirected graph*. A node $N_2$ is a *neighbor* of $N_1$ if the nodes are connected by an edge: $\{N_1, N_2\} \in E$. If the robot can travel from node $N_1$ to node $N_2$ but not in the opposite direction, the resulting collection is called *directed graph* where nodes can be in- and outneighbors. See Figure 1 for the distinction of directed and undirected graphs [5].

A path in $G$ is a sequence of nodes $\{N_1, N_2, \ldots, N_n\}$, s.t. $\{N_{i-1}, N_i\} \in E$. A graph is *connected* if there exists a path connecting nodes $N_i$ and $N_j$ for all nodes $N_i$ and $N_j$ in the graph. The solution to the path planning problem is the path connecting the start node with the goal node [5].

A special type of graph is a *tree*. It has a special node called *root*, the only node without an incoming edge (no parent node). Nodes without further child nodes are called *leaves*.

There are three basic methods to search a graph - be it a tree, a grid or any other graph structure - for a desired node, typically the goal node: *Depth-first search*, *breadth-first search* and *best-first search* which will be discussed in the following section [5].

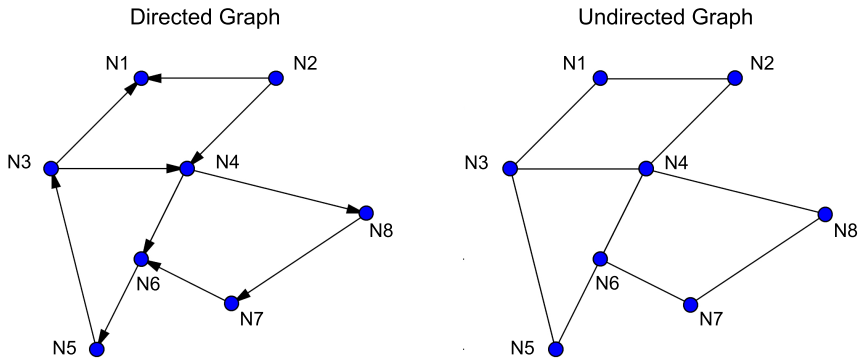


Figure 1: An example for a directed graph (left) and an undirected graph (right).

## 2.2 Uninformed and Informed Search

*Uninformed* search algorithms rely solely on the structure of the underlying graph and the neighboring information to retrieve a path; no additional knowledge is incorporated. Examples for uninformed algorithms include the *depth-first search* and *breadth-first search*.

In contrast, *informed* search algorithms exploit heuristic information such as the remaining distance to the goal to be more efficient. The *best-first search* is an example for informed algorithms [7].

### Depth-first search

The depth-first search starts at the initial node $N_{init}$ and explores the first branch until the goal node or the end of the branch is reached. If the algorithm reaches the end of a branch, it backs up and explores the next branch and so forth until reaching the goal node $N_{goal}$. The algorithm stores all nodes to explore in a last-in first-out list (stack); a new node is immediately explored before trying already stored nodes. Figure 2 (left) shows the order in which nodes are expanded using the depth-first algorithm [5].

### Breadth-first search

The breadth-first search starts at the initial node $N_{init}$ and explores all children of that node first. If the goal node $N_{goal}$ was not yet found, it explores all children of the first child node and so on until $N_{goal}$ is found.

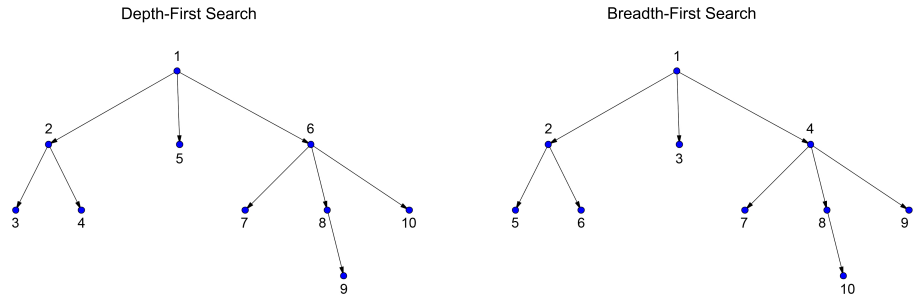


Figure 2: The search tree on the left hand illustrates the order in which the nodes are expanded in a depth-first search, the tree on the right hand side shows the order of a breadth-first search.

The search tree is explored layer by layer. The nodes are stored in a first-in first-out list (queue); a new option is added at the end of the list and all already stored nodes are explored first. Figure 2 (right) reflects the method of the breadth-first algorithm [5].

Note that the depth-first algorithm favors nodes closer to the goal node whereas breadth-first search favors those close to the initial node.

#### Best-first search

Instead of a simple stack or queue, the best-first search makes use of a sorted list which requires heuristic information about the nodes. Incorporating additional knowledge to a search algorithm increases its performance, as more promising nodes are chosen first. To estimate the "promise" of a node, an evaluation function $f(n)$ is applied to every node. The function can depend on characteristics of the node itself, information about the path found so far, information about the distance to the goal and any other prior knowledge relating to the problem. A best-first search algorithm generates a path by always choosing the node with the lowest value for $f(n)$ as next step [22]. The most prominent example for a best-first search algorithm is *A**.

### 2.3 A* - The Standard Admissible Algorithm

The A* algorithm was introduced by Hart, Nilsson and Raphael in 1968 [11] as a combination of two then common path finding approaches: the *formal* and the *heuristic* approach. While formal algorithms usually guarantee to

find a shortest path, heuristic algorithms are said to be greedy; they "guide" themselves toward the goal. The applied heuristic function estimates the remaining distance to the goal and is used as weighting function [11].

### 2.3.1 Characteristics of A*

A* operates on a directed graph with costs assigned to each edge. Based on these costs, A* assigns an evaluation function $f(n)$ to every node $n$. $f(n)$ is the lower bound on the costs for the optimal path through $n$.

$$f(n) = g(n) + h(n) \tag{1}$$

$g(n)$ represents the cost for moving from the start node $s$ to the current node $n$ by summing up the cost of each traversed edge. $h(n)$ is the estimated cost for moving from $n$ to the goal node $g$. This cost is calculated based on a distance function that relates to the current problem. A basic distance function is the Euclidean distance (Equation 2) which calculates the straight-line distance between two points $A$ and $B$ [6].

$$dist(A, B) := \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2} \tag{2}$$

When planning on 4-connected grids where robots are constrained to only move horizontally and vertically, the Manhattan distance can be calculated instead (Equation 3). The Manhattan distance is the sum of the absolute differences between the $x-$ and $y-$ coordinates of two points $A$ and $B$ [6].

$$dist(A, B) := |A_x - B_x| + |A_y - B_y|. \tag{3}$$

Both distance functions are calculated regardless of any obstacles and need to be weighted with the step cost that is assigned to the corresponding edges.

The choice of the heuristic function impacts the efficiency of the A* search. A "bad" heuristic will take more time than possibly needed to compute a solution and not find an optimal path. Provided an *admissible* and *locally consistent* heuristic, A* is *optimal*. An optimal search algorithm is guaranteed to find a path with lowest cost possible.

A heuristic function $h(n)$ is admissible (optimistic) if and only if:

$$\forall N \in G\colon\ 0 \leq h(N) \leq h'(N) \tag{4}$$

$h(N)$ is the heuristic estimation of the cost to the goal, while $h'(N)$ is the actual cost [17]. This means that if the heuristic for the current node to

the goal node never overestimates the actual cost, the search algorithm is admissible.

Another way to characterize a heuristic function is monotony. A heuristic is monotone or locally consistent if for every pair of adjacent nodes $N$ and $N'$ the following equation is satisfied:

$$\forall N, N' \in G: \; 0 \leq h(N) \leq h(N') + k(N, N') \tag{5}$$

$k(N, N')$ is the cost of getting from node $N$ to node $N'$. The Manhattan distance and the Euclidean distance both are examples for admissible and locally consistent heuristics.

Besides optimality, completeness is another important characteristic for a path planning algorithm. An algorithm is *complete* if in finite time it either finds a path or correctly determines that no solution exists. For a detailed proof of A*'s optimality and completeness, see [11].

### 2.3.2 A Short Example of A*

The following example of the A* algorithm is inspired by [21]. For an illustration of the complete series of planning steps, see Appendix **??**.

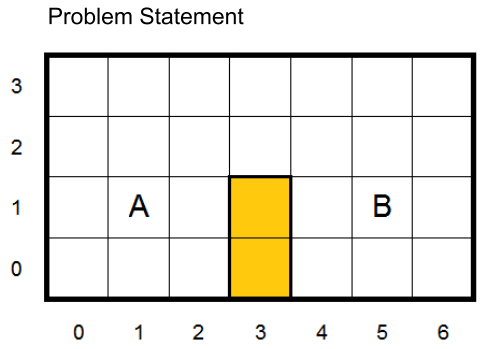


Figure 3: Grid representation of a world with a start position $A$, a goal position $B$ and an obstacle (highlighted in yellow).

Figure 3 represents the problem statement: The intention is to move along the shortest path from the start point $A$ to the goal point $B$ by avoiding the obstacles (highlighted in yellow) which could be simply thought of as walls. The search area is simplified to a grid thus reducing it to a two-dimensional array. The status of every cell is set to either be free or occupied,

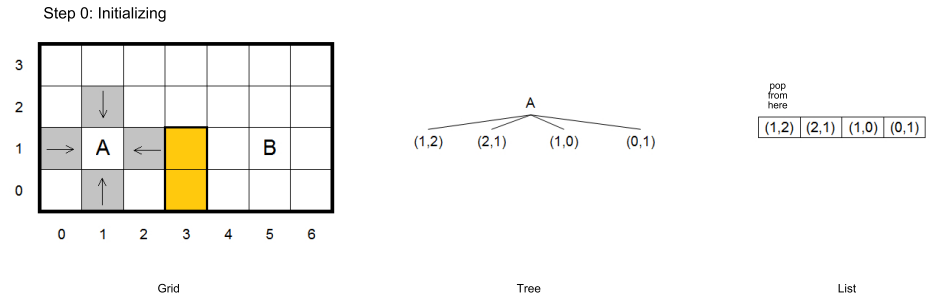


Figure 4: Grid representing the initial state (left), tree representing the parent-child analogy (center), open list representation (right).

where occupied refers to containing an obstacle. As a further simplification, it is assumed that the robot is constrained to move vertically or horizontally; diagonal moves are illegal. Such a graph is called 4-connected grid, as every node has four neighbors.

### Starting the Search

As mentioned above, the A* algorithm is initialized by adding the start node $A$ to the open list. To start the search, $A$ is taken from the open list, added to the closed list and expanded. This adds the four neighboring nodes $(1, 2)$, $(2, 1)$, $(1, 0)$ and $(0, 1)$ to the open list. For each neighboring node, $A$ is stored as parent node. To finally retrace the path, A* spans a search tree consisting of all explored nodes representing their parent-child relationship.

Figure 4 shows the grid (left) after this first expansion: The nodes currently in the open list are highlighted and an arrow points to the parent node. The tree representation in the middle shows the parent-child analogy between the nodes and on the right hand side, the open list and the nodes it currently contains are illustrated.

### Evaluating the Costs

Next, the costs for each node need to be assessed. In this example, a cost of 1 is assigned to every edge. As A* is operated on a 4-connected grid, the Manhattan distance is used to calculate the heuristic function $h(n)$ for every node. This means to determine the total number of horizontal and vertical moves necessary to get from the current node to the goal node, regardless

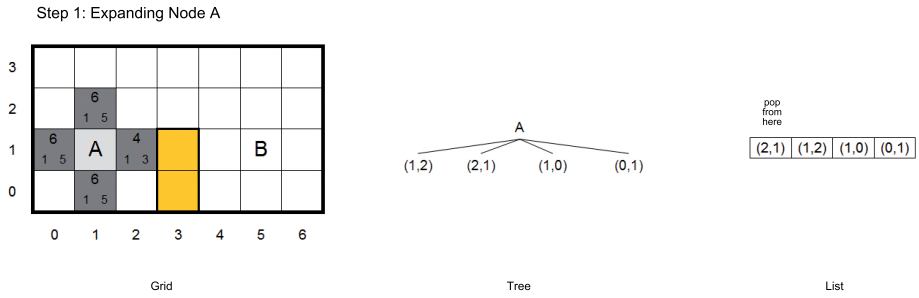


Figure 5: In the grid (left), for each cell the values of $f(n)$ (top), $g(n)$ (bottom left) and $h(n)$ (bottom right) have been added. The open list (right) is displayed in sorted order.

whether a node is free or occupied and weighting it with the edge cost. $g(n)$ is calculated by summing up the number of steps needed to reach the current node from the start node and weighting the sum with the edge cost.

Having assigned the $f(n)$-values to every node, A* continues by chosing the most promising node of the open list as next step. In Figure 5, the value of the cost function has been added to each node in the grid (top value in every cell). Additionally, the values for $g(n)$ and $h(n)$ were also inserted (values at the bottom left and right in each cell, respectively). The open list is sorted according to increasing $f(n)$-values.

The most promising node at this point is node $(2, 1)$ with an $f(n)$-value of 4. It is deleted from the open list and added to the closed list. All adjacent nodes to $(2, 1)$ not being obstacles are added to the open list. Thus, node $(3, 1)$ is ignored, but nodes $(2, 2)$ and $(2, 0)$ are added to the open list and their parent nodes are set. The start node $A$ is already in the closed list and a path via $(2, 1)$ to $A$ would result in higher costs such that $A$ remains in the closed list. Nodes already in the open list are evaluated whether the new path has lower cost. If this is the case, their values for $g(n)$ and their backpointers would need to be updated. The nodes in the open list do not need to be updated in this example.

### Breaking Ties

Every node in the open list has the same value for $f(n)$ now. In this case, A* can be implemented to either chose an arbitrary node or to incorporate a rule for breaking ties. One possibility for tie breaking is to favor nodes that are closer to the goal node, assuming that a path via this node would

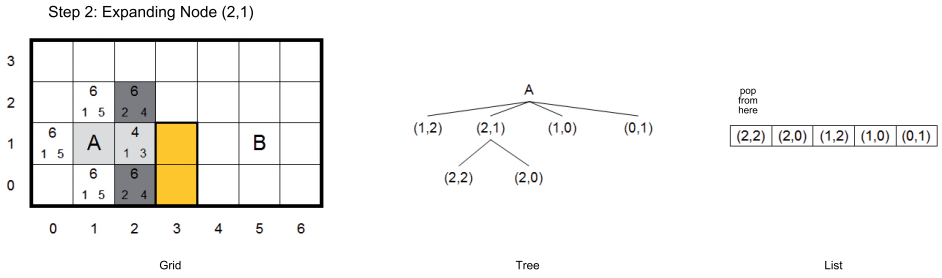


Figure 6: The expansion of node $(2, 1)$ is shown in the grid (left). The new child nodes are added (center) and the open list is updated accordingly (right).

be shorter than others. Thus, all nodes in the open list with the same $f(n)$-value are further sorted by $h(n)$-values. The sorting order in the open list (right) in Figure 6 already reflects this tie breaking rule.

Figure 7 illustrates the expansion of the next node: The most promising node $(2, 2)$ is removed from the open list and added to the closed list. Two new adjacent nodes are added to the open list and the tree representation is updated. The neighboring node $(1, 2)$ does not need to be updated, as the path via node $(2, 2)$ would increase the cost.

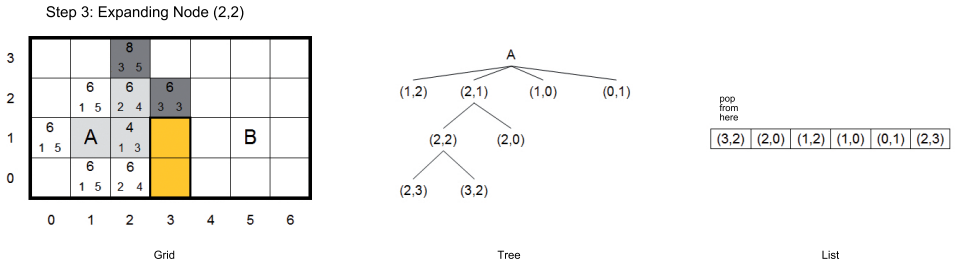


Figure 7: Illustration of the expansion of node $(2, 2)$.

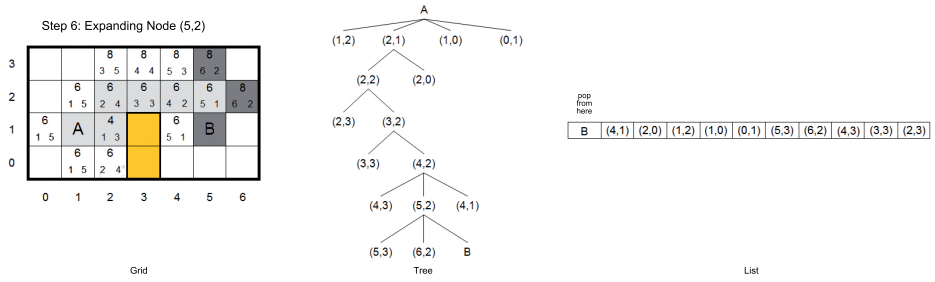


Figure 8: When expanding node $(5, 2)$, the goal node is added to the open list.

## Completing the Path

To compute the complete path, the process explained above needs to be repeated until the goal node is added to the closed list. Figure 8 shows the grid (left) one step earlier, when node $B$ is the next node in the open list. After popping the goal node from the open list and adding it to the closed list, A* follows the backpointers to the parent nodes which eventually leads back to node $A$. The complete path from the goal node to the start node can be determined, as illustrated in Figure 9. This path represents the lowest cost-path to get from $A$ to $B$.

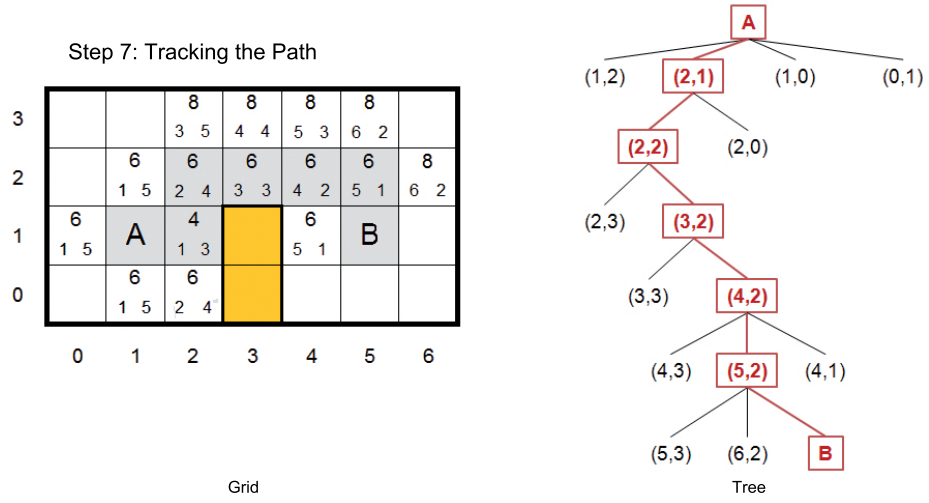


Figure 9: Tracking the path from the goal node to the start node by visiting each node's parent.

# 3 From Single- to Multirobot Systems

The single robot path planning problem is usually effectively solved running A* on a graph. However, when solving the multirobot path planning problem for a composite robot in a composite configuration space with A*, limits are soon reached: The computational complexity increases with every additional robot. This allows for a variety of multi-robot path planners to be researched and implemented.

The benefits of multirobot systems are obvious: Multiple robots can carry out multiple tasks, which is essential in industrial assembly or search and rescue. Additionally, multiple robots can guarantee sensor coverage for tasks such as exploration or surveillance. Moreover, having more than one robot in use increases the robustness and the fault tolerance of the system. Even if a sensor fails or one robot is out of communication range, redundancy assures that the system still functions [20].

## 3.1 Curse of Dimensionality

Despite the obvious advantages of multirobot systems, a major drawback arises: The size of the joint configuration space increases exponentially with every additional robot, thus eventually leading to computational infeasibility. This phenomenon is generally referred to as *curse of dimensionality*, a term coined by Richard Bellman in 1957 [1]. Bellman pointed out that computational tasks carried out in higher dimensions are much harder to solve than in lower dimensions.

For a better understanding, the problem can be illustrated as follows: Within a $d$-dimensional "cube" $c_1$ with edges of length $l_1 = 1$, a smaller cube $c_2$ with an edge length of $l_2 = \frac{1}{2}$ is placed. The edges of the smaller cube are positioned along the edges of the bigger one. With $d = 1$, the "cube" is just a line segment, thus the volume of $c_2$ is half of the volume of $c_1$. If $d = 2$, the cubes are squares and their volumes' ratio is $\frac{1}{4}$. In $d = 3$, cube $c_2$ fills a volume of $\frac{1}{8}$ of $c_1$ and so on and so forth. For a dimension of $d = 10$, the smaller cube - still with half the edge length of the bigger one - only fills about a thousandth of the total volume. Thus, as the space occupied by $c_2$ within $c_1$ decreases continuously, it gets harder to spot that cube within $c_1$. Figure 10 illustrates this notion [12].

Analogously, it becomes more difficult to "find" or compute a path from a start to a goal location within a given configuration space that grows exponentially with every additional robot.

Assuming two dimensional robots that store each of their coordinates

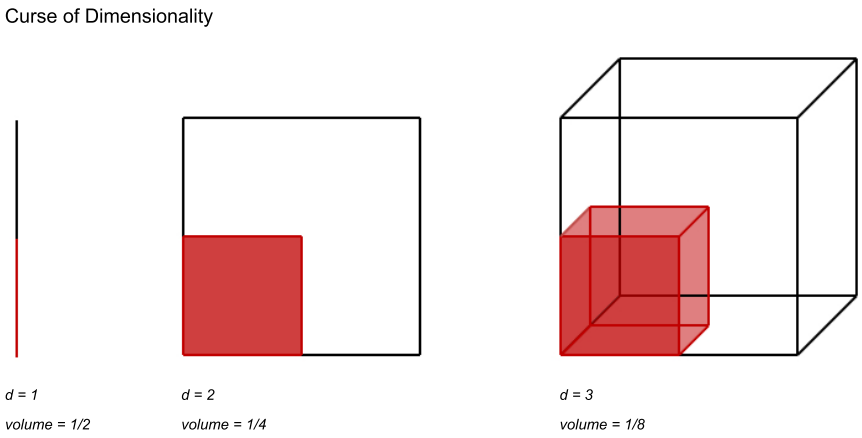


Figure 10: The problem of the "Curse of Dimensionality" illustrated for a one-, two- and three-dimensional "cube".

in a single byte, $2n$ bytes are needed to represent a single configuration of $n$ robots. At each step, A* expands $5^n - 1$ neighbors of a given node which requires $2n(5^n - 1)$ bytes of memory. Every expansion for a single robot system thus requires 8 bytes, whereas a system with 12 robots requires approximately 6GB of memory size for a single expansion.

# 4 Path Planning for Multiple Robots

There is much extant research in developing algorithms to solve the multi-robot path planning problem. However, the approaches to handle the curse of dimensionality differ: While some algorithms search the joint configuration space[1] of all robots and thus guarantee optimality and completeness, others sacrifice optimality for computational feasibility by searching lower dimensional sub-spaces. To reflect these contrasting concepts, multirobot path planning algorithms can be further divided into two main categories: *coupled* and *decoupled* algorithms [19].

## 4.1 Decoupled Planning Algorithms

Decoupled algorithms solve the multirobot path planning problem by planning a path for each individual robot separately; possible conflicts are resolved afterwards. As a result, the complexity of the system is reduced by searching lower dimensional subspaces. Therefore, decoupled algorithms typically produce results more quickly but optimality or even the existence of a solution can not be assured [17].

Examples for decoupled algorithms include a velocity planner proposed by Kant and Zucker [13]. The velocity planner starts by planning a path for each robot individually and resolves possible conflicts by adapting the velocity of the robots involved. A variant of the velocity planner called Incremental Coordination was developed by Saha and Isto [23]. Their algorithm combines the two steps by adapting the velocity of the current robot to be planned a path to all other robots that have already been assigned a path.

### 4.1.1 Priority Planning

Another example for a decoupled multirobot planning approach is the priority planner as first described by Erdmann and Lozano-Perez [8]. Instead of planning paths for all robots in a high dimensional search space, paths are planned one robot at a time in descending order of priority. There are several methods to assign priorities to robots, i.e. based on the individual path length [2] or on favoring straight-line motions [4]. Robots can also be considered in a random but fixed order.

The main characteristic of a priority planning algorithm is that a path for one robot not only needs to avoid static obstacles but also paths of robots

[1] The joint configuration space is the Cartesian product of the robots' individual configuration spaces.

with higher priorities. Those robots are considered *dynamic obstacles* and have already been assigned a path. Paths of higher priority robots will not be changed again. To compute a path that does not conflict with dynamic obstacles, the planner needs to have a notion of time. Thus, in addition to the coordinates of each node in a path, the corresponding timestep is captured.

Unfortunately, a vanilla priority planner may cause deadlocks and not succeed in computing a solution, even if one exists. It is neither complete nor optimal.

## 4.2 Coupled Planning Algorithms

Coupled algorithms aim at finding solutions for all robots in their joint configuration space, thus guaranteeing that an optimal path will be found if one exists. However, these algorithms are only applicable for a small number of robots as their computational complexity increases exponentially with every additional robot [17].

Combining multiple robots to one complex robot allows the path planning problem to be solved using a single robot algorithm like A*. Variants of A* such as Iterative-deepening A* (IDA*) [15] or Hierarchical Cooperative A* (HCA*) [26] are designed for solving the multirobot path planning problem. In this thesis, the focus lies on a technique called *Operator Decomposition* propsed by Standley [27] as it is part of the new algorithm OD-M*.

### 4.2.1 Operator Decomposition

Operator Decomposition (OD) [27] is a variant of A* and offers an improvement to the exhaustive node expansion of A*. OD constructs neighbors of a single node incrementally. Doing so delays the instantiation of nodes that are not optimal according to path cost.

Standley's idea is to operate the algorithm in a decomposed time space. To approach one timestep, each robot is considered one after the other, assigning moves to every robot sequentially. As long as not all of the robots have been assigned a move, nodes are thought of as *intermediate* nodes. *Standard* nodes are nodes with a move assignment for all robots for the current timestep.

Expanding the standard start node for the first robot $r^1$ results in a first set of intermediate nodes; one for every possible move for $r^1$. The resulting intermediate nodes are put on the open list, sorted by increasing cost. The

algorithm continues with robot $r^2$ based on the lowest cost node where robot $r^1$ has already been assigned a move. Collisions with the new position of robot $r^1$ have to be avoided, while other robots that have not yet been assigned a move are ignored. The next series of intermediate nodes consists of one node for every possible move for $r^2$.

Figure 11 illustrates the expansion of the intermediate nodes for a problem including two robots in detail. Robot $r^1$ is supposed to move from position $A1$ to position $B1$ and robot $r^2$ from $C1$ to $C0$. The standard start node is $\{A1, C1\}$ and unit cost is assigned to each edge. Thus, the optimal path cost is $f = 2$, as each robot requires one step to reach its goal position.

First, intermediate nodes are generated based on the possible moves for $r^1$. Including a wait option, this results in four intermediate nodes, with node $\{A2, C1\}$ having minimum cost. The optimal node is taken from the open list to proceed with the expansion for robot $r^2$. To avoid collision with $r^1$, the node $\{B1, B1\}$ is not expanded. The most promising node in the open list is now $\{B1, C0\}$, the goal node.

Operator Decomposition has generated seven nodes in this example. Solving this problem with A* would require the expansion of 15 nodes $(4 \times 4 - 1)$. OD scales better with more complex problems and reduces computational costs compared to A*. Recently, Standley proposed variants of Operator Decomposition, including an any-time algorithm [28].

### 4.3 Dynamically Coupled Algorithms

Recently, a new category of algorithms has emerged that overcomes the disadvantages of both the coupled and decoupled approaches by dynamically coupling robots that are in collision. Examples include a velocity planner from Krishna et al. [16]. To avoid collisions, robots try to independently adapt their velocity. If this attempt fails, colliding robots start to cooperate to find safe velocity profiles. If this still does not succeed, the cooperation is extended to uninvolved robots to find velocity profiles that guarantee a solution.

The planning time algorithm from Van den Berg et. al. [3] seeks to minimize the size of the largest coupled group of robots necessary to ensure that a solution is found. Robots are forced to move sequentially - before or after each other. If these move constraints cause cycles, it is an indication to treat the involved set of robots as coupled group.

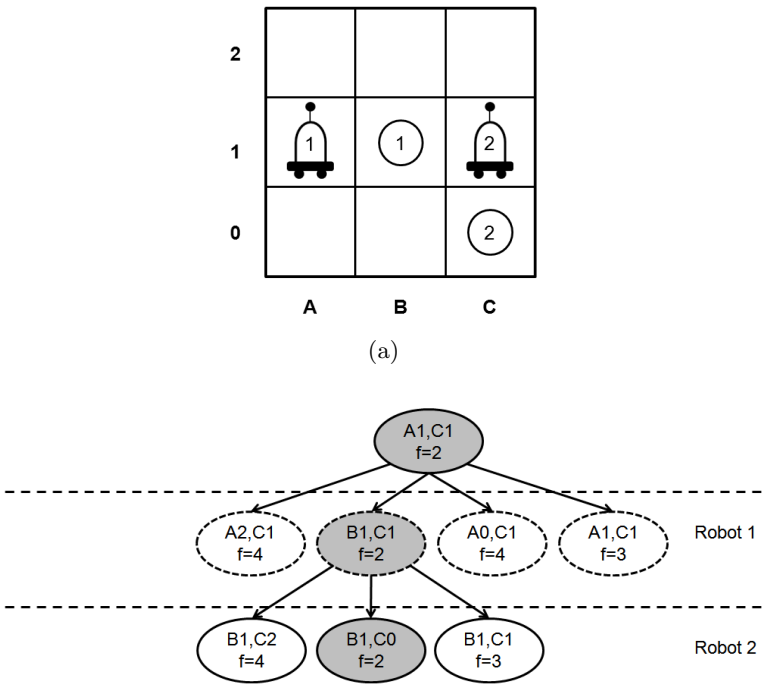


Figure 11: **(a)** A simple path planning problem including two robots. Nodes $A1$ and $C1$ represent the start nodes, $B1$ and $C0$ the goal nodes. **(b)** $\{A1, C1\}$ is the root node of the path planning problem. Intermediate nodes are symbolized by dashed circles, standard nodes by solid circles. With unit costs assigned to each edge, the cost for the optimal solution is $f = 2$. After expanding the nodes for the first robot, OD continues by expanding successor nodes for the intermediate node with optimal cost - $\{A2, C1\}$. When expanding nodes for the second robot, collisions with the new position of the first robot have to be avoided. Eventually, node $\{B1, C0\}$ has lowest cost. As it is the goal node, the planning problem is solved. [25, adapted]

### 4.3.1 Subdimensional Expansion

Subdimensional Expansion was proposed by Wagner in his previous work [29] as an algorithm that dynamically constructs a search space: Initially, an individually optimal path is planned in each robot's individual configuration space. The individually optimal paths form a one-dimensional search space embedded in the joint configuration space of all robots. A planning algorithm explores this search space to detect robot-robot collisions. If robot-robot collisions are detected, the dimensionality of the search space is locally

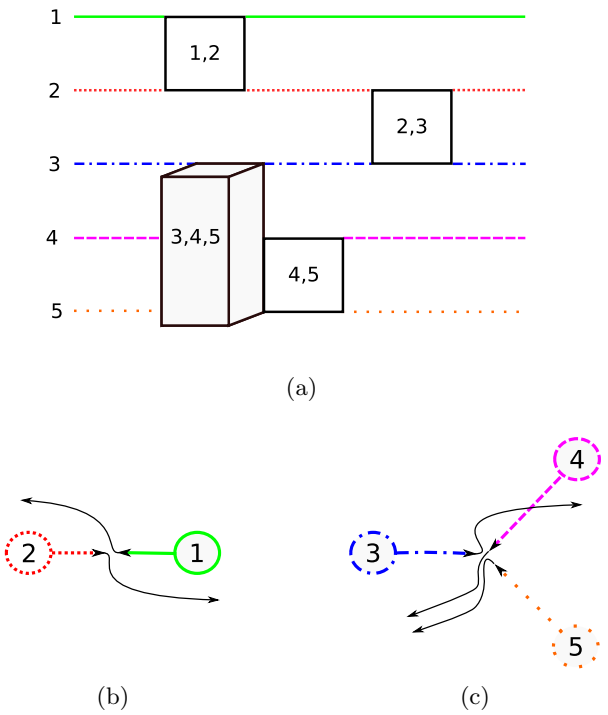


Figure 12: **(a)** A conceptual visualization of a search space of variable dimensionality for five robots. Initially, each robot is restricted to follow its individual policy, as represented by a single line. **(b)** When robots 1 and 2 are found to collide, the dimensionality of the search space must be locally increased, which is represented by a square. **(c)** When three robots collide while being constrained to their individual policy, the local dimensionality of the search space must be further increased, to include all local paths of the three robots. This is represented by the cube. Robot 3 does not need to be coupled with robots 4 and 5 any longer, once it clears their way, even though robots 4 and 5 continue to interact. [29]

augmented (Figure 12). Thus, the dimensionality of the search space is always kept as low as possible.

The two fundamental concepts of Subdimensional Expansion are the *individual policy* and the *collision set*. The individual optimal policy of the $i$-th robot $r^i$ maps the position of $r^i$ in its configuration space to its optimal action at that position in the absence of other robots. Obeying its individual policy at any position produces an individually optimal path for that robot.

If following the indiviual policies leads to a robot-robot collision at a specific position, those robots are stored in the corresponding collision set. The collision set $C_k$ for a given position $q_k$ in the joint configuration space

is the set of robots $r^i$ for which the planner has found a path through $q_k$ to a collision between $r^i$ and another robot. When the planning algorithm extends a path from $q_k$, every robot not in $C_k$ obeys its individual policy, while all possible actions must be considered for robots in $C_k$. Thus, the search space of variable dimensionality is implicitly defined.

A *backpropagation set*, consisting of all neighbors of node $n_k$ through which the path planning algorithm has found a path to $n_k$, is used to keep the collision sets updated. When a robot-robot collision is found, backpropagation means that for each neighboring node $n_l$ the collision set $C_k$ is added to $C_l$. Next, $C_l$ is added to the collision set of each node in the backpropagation set of $n_l$ and so forth.

### M* - Subdimensional Expansion with A*

The M* algorithm is an implementation of Subdimensional Expansion for graph search [29]. The configuration space of robot $r^i$ is represented as directed graph and A* is used as the underlying path planning algorithm.

To restric A* to the low dimensional search space generated by Subdimensional Expansion, M* only expands the *limited neighbors* of a node $n_k$. The limited neighbors of $n_k$ are the subset of neighbors of $n_k$ which can be reached if every robot $r^i \notin C_k$ obeys its individual policy at node $n_k$.

Recursive M* (rM*) is a variant of M* which improves the way physically separated sets of colliding robots are handled. While basic M* couples the planning for all such sets, rM* can plan for each disjoint subset separately. The computational cost is then exponential in the size of the largest set of colliding robots rather than the total number of colliding robots.

Wagner has shown that Subdimensional Expansion can also be coupled with probabilistic planners such as rapidly-exploring random trees (RRTs) [18] and probabilistic roadmaps (PRMs) [14] resulting in the algorithms sRRT and sPRM, respectively [30].

# 5 OD-M* and Independence Detection: A New Optimal Algorithm for Multirobot Path Planning

As contribution to the multirobot path planning problem, *OD-M**, a new optimal multirobot path planning algorithm, is presented. OD-M* combines Subdimensional Expansion (M*), that dynamically constructs the search space, with the coupled planning algorithm Operator Decomposition. Moreover, a variant of OD-M* is proposed, where OD-M* is embedded into the planning framework Independence Detection (ID) [27]. ID minimizes the necessary coupling of robots. This variant is called *ID with OD-M**.

In short, ID with OD-M* works as follows: Initially, an individually optimal path is planned for every robot. If robots are found to be in collision when following their initial optimal paths, the Independence Detection framework groups those robots. Subdimensional Expansion then generates search spaces of minimal dimensionality for every group of robots that needs to be assigned new paths. Finally, Operator Decomposition computes optimal paths for each group within the search space generated by Subdimensional Expansion.

## 5.1 ID with OD-M* in Detail

The highest level of planning of the new algorithm *ID with OD-M** is the Independence Detection framework which decomposes the coupled planning problem into a set of independent subproblems as follows:

Initially, every robot $r^i$ is assigned to an individual group $g_j$ and optimal paths are planned for each group separately, ignoring the existance of other robots. The execution of the paths is then simulated for all singleton groups. When collisions between two groups are detected, an attempt is made to find new optimal collision free paths for the first group. The replanning is based on the concept of a priority planner: The robots in the other group are treated as dynamic obstacles that need to be avoided. If the attempt to find alternate optimal paths for the first group fails, the paths for the robots in the second group are replanned, with robots in the first group treated as dynamic obstacles.

For the replanning of singleton groups, a vanilla priority planner is called. The coupled planner OD-M* is called whenever replanning for a group of more than one robot is necessary. To guarantee optimality, only nodes that have equal cost as the initial path are allowed to be put onto the open list. If

both replanning attempts fail, the two groups are merged and new optimal paths are found using OD-M* for the combined group.

In order to minimize future replans, the new paths should result in as few conflicts with robots not in the merged group as possible. Thus, Independence Detection would require the number of collisions with "out-of-group" robots to be used for tie breaking. The current implementation of ID with OD-M* does not include this tie breaking rule and computes paths for the merged group regardless of other robots. This behaviour is being implemented and adds additional value to the algorithm (For more details see the Future Work section, Chapter 7.1).

Groups are also merged if they are found to be in collision for the second time. This behaviour is needed to avoid infinite loops where replanning $g_1$ to avoid $g_2$ results in a conflict with $g_3$ and replanning $g_1$ to avoid $g_3$ would result in a conflict with $g_2$ again. Each group is required to keep track of other groups with which it has been in collision to prevent that oscillation.

The computational complexity of ID is exponential in the size of the largest group rather than the overall number of colliding robots. See Algorithm 1 for a complete pseudo-code example of the algorithm ID with OD-M*.

For the simulations in this thesis, Independence Detection with OD-rM* was used instead of ID with OD-M*. This algorithm differs mostly in how paths are replanned for coupled groups: When multiple sets of robots are in collision within one group, rM* can calculate paths for these subsets separately and thus further reduces the complexity of the path planning problem.

**Algorithm 1** Pseudocode for ID with OD-M* for $n$ robots:
$i$ - robot indices; $i \in I = \{1, \ldots, n\}$
$r^i$ - robots
*based on [27]*

```
paths ← ∅
groups ← ∅
for all i
    groups(i) ← i
    groups(i).collisions ← ∅
    paths(i) ← find_optimal_path(r^i)
# Iterate over all groups; suppose collision between groups g_a and g_b is detected
while collision
    # (1) Groups g_a and g_b have been in collision before
    if g_b ∈ g_a.collisions
        new_group ← merge(g_a, g_b)
        update groups
        new_paths ← OD-M*(new_group)
        if ¬failure
            paths.update(new_paths)
            break
    # (2) Replan paths for group g_a
    # Paths of group g_b are passed as constraint
    new_paths ← priority_replanning(g_a)
    if ¬failure
        paths.update(new_paths)
        update groups.collisions
        break
    # (3) Replan paths for group g_b
    # Paths of group g_a are passed as constraint
    new_paths ← priority_replanning(g_b)
    if ¬failure
        paths.update(new_paths)
        update groups.collisions
        break
    # (4) No solution found thus far; join groups
    new_group ← merge(g_a, g_b)
    update groups
    new_paths ← OD-M*(new_group)
    if ¬failure
        paths.update(new_paths)
        break
return paths
```

# 6 Simulation Results

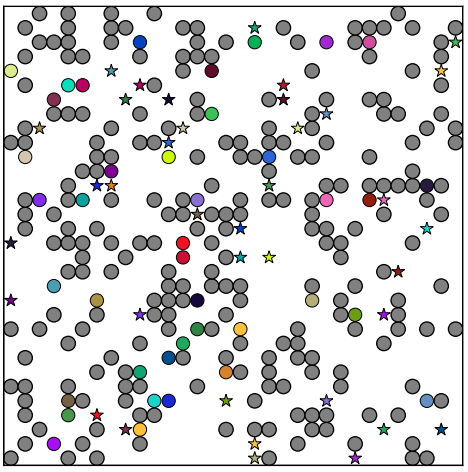

Figure 13: A typical 8-connected grid world with 32x32 cells for a test run including 40 robots. Colored cirles represent initial positions of the robots, colored stars their goal positions. Gray circles represent the obstacles. For each number of robots, 100 randomly generated worlds were tested.

To test the performance of OD-rM*, simulations were run on a Core i5-2500 computer at 3.30 GHz (Turbo mode disabled) with 8 GB of RAM. All simulations were implemented in unoptimized python. As environment, a fixed-size, 8-connected grid of 32x32 cells and a probability of 20% for each cell being an obstacle was used as proposed by Standley [27]. Initial and goal positions were chosen randomly, but it was assured that there existed a path from a robot's initial to its goal position (Figure 13).

At most 5 minutes were allowed for each trial to find a solution. For each number of robots, 100 random environments were tested. The percentage of trials that were successful within 5 minutes as well as the time required to find solutions by the 10'th, the 50'th and the 90'th percentile of trials was recorded. The run times are plotted on an exponential scale such that exponential growth would appear as a straight line.

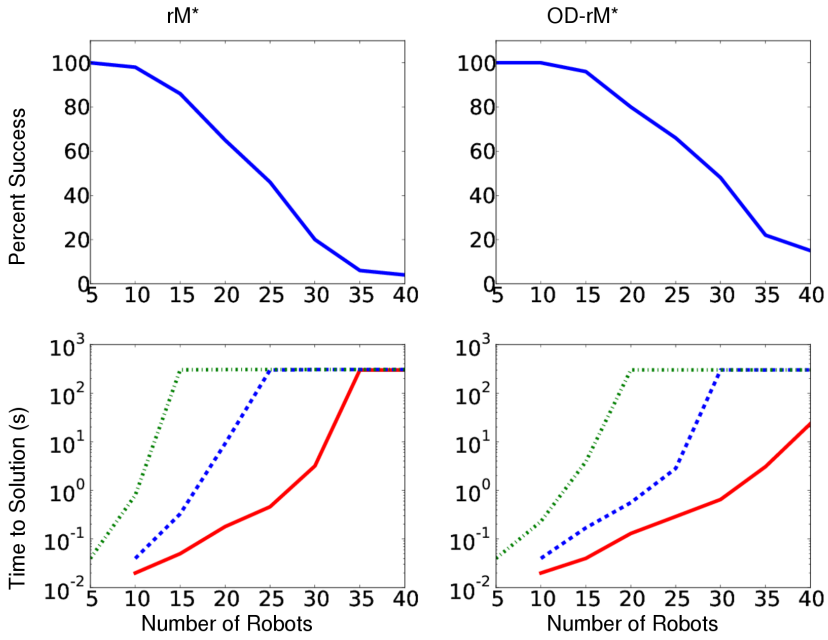


Figure 14: Results for basic rM* and OD-rM*. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes. The bottom graphs show the 10'th (red solid line), 50'th (blue dashed line) and 90'th (green dotted line) percentile of times required to compute the solution. Both algorithms were simulated for 5 to 40 robots, increasing the robot number by 5 for every new set of 100 trials.

### 6.1 rM* vs. OD-rM*

The first set of trials compares the performance of basic rM* to that of OD-rM*. As rM* uses A* as underlying path planning algorithm, OD-rM* typically expands less nodes than rM*. Thus, OD-rM* was expected to solve more instances within the given time limit. As shown in Figure 14, basic rM* solves about 10% less trials than OD-rM* for problems including up to 20 or more than 35 robots. For problems involving 25 robots, OD-rM* outperforms rM* by about 20% and even by 30% for 30-robot-problems.

As OD-rM* generates less nodes than rM*, also the runtime of the algorithm is improved as depicted in the time plots in Figure 14. While even the 10'th percentile reaches the time limit for rM*, 10% of the trials with 40 robots are solved in under 25 seconds with OD-rM*.

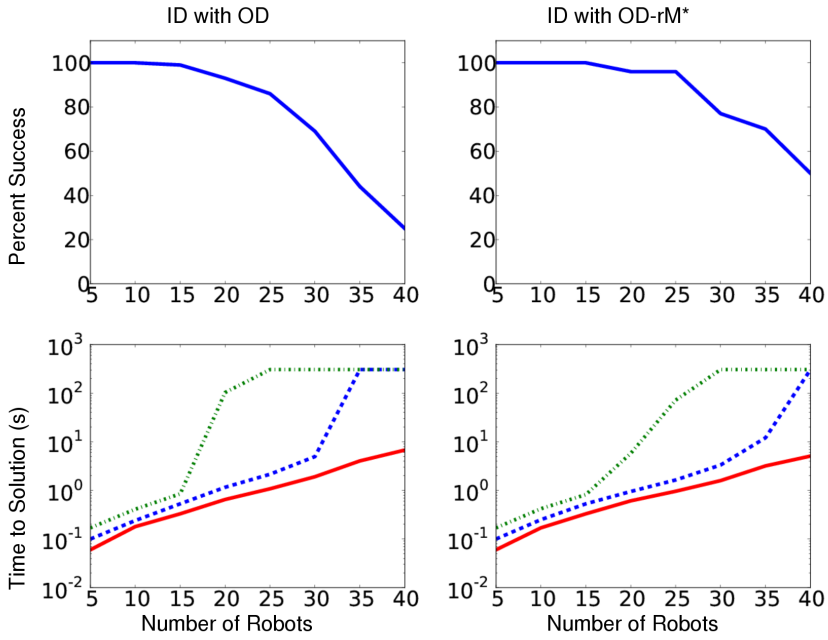


Figure 15: Results for ID with OD and ID with OD-rM*. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes. The bottom graphs show the 10'th (red solid line), 50'th (blue dashed line) and 90'th (green dotted line) percentile of times required to compute the solution. Both algorithms were simulated for 5 to 40 robots, increasing the robot number by 5 for every new set of 100 trials.

Reaching the time limit of 5 minutes results in the plateauing obvious in the time plots. Note the missing values for the 10'th and 50'th percentile in the graphs for instances of 5 robots for both algorithms. The reason is the accuracy of the timing function of one millisecond. Instances solved in less than one millisecond are automatically set to zero which cannot be displayed in the exponential scale.

## 6.2 ID with OD vs. ID with OD-rM*

The next set of trials was ment to oppose the performance of Independence Detection with basic OD to that of ID with OD-rM*. While both algorithms solve about the same number of instances for problems involving up to 15 robots, ID with OD-rM* outperforms ID with OD by approximately 10% for

problems including 25 or 30 robots. Additionally, ID with OD-rM* solves 25% more of the problems that include 35 or 40 robots than ID with OD. At 40 robots, ID with OD-rM* has twice the success rate of ID with OD.

The reason for the good performance of OD-rM* is that rM* restricts all robots not in collision to their individual optimal policy when replanning groups. A replan based on OD is only necessary for robots in collision. The dimensionality of the search space and thus the complexity of the replanning problem is much lower than with basic OD. The Operator Decomposition algorithm needs to expand nodes for every robot in the group in the joint configuration space leading to a higher runtime and a lower success rate. The results for ID with OD and ID with OD-rM*, respectively, are shown in Figure 15.

ID with OD-rM* shows a performance increase of about 30% for problems including 40 robots, compared to basic OD-rM* (Figure 14). While OD-rM* solves about 20% of trials, ID with OD-rM* solves 50% of these instances. The 90'th percentile reaches the time limit already for problems involving 20 robots with OD-rM*, whereas ID with OD-rM* solves these problems within 10 seconds for the 90'th percentile. The additional level of coupling that comes with Independence Detection allows the OD-rM* algorithm to solve smaller subproblems compared to basic OD-rM*. The subproblems thus have a lower dimensionality and require less computation time.

# 7 Conclusion

OD-M* and ID with OD-M*, respectively, are new optimal planning algorithms to solve the multirobot path planning problem. The algorithms address the exponential growth in both the size of the configuration space and the number of nodes generated by the basic A* algorithm.

If Independence Detection is applied as framework for the OD-M* algorithm, robots found to be in collision when following their optimal paths are combined into one group. The search algorithm is then applied on smaller subsets of robots rather than the total system; the size of the largest group determines the complexity of the problem.

M* (or rM*) constructs low-dimensional search spaces instead of exploring the joint configuration space of the multirobot system. Whenever robots are in collision, the dimensionality of the search space is locally increased.

Operator Decomposition significantly reduces the number of nodes A* would generate at every step by assigning moves to every robot sequentially within each timestep and by favoring low-cost nodes.

As a result, ID with OD-rM* outperforms the optimal coupled algorithm OD as well as M*. ID with OD-rM* overcomes the disadvantages of most coupled and decoupled algorithms: It is a multirobot path planning approach that is both optimal - it produces paths of lowest costs possible - and computationally feasible. Results have shown that ID with OD-rM* is approximately two times faster than ID with OD for problems including 25 robots and solves about 12 times more instances at 40 robots than basic rM*. Thus, the new planning algorithm is suitable for various multirobot applications such as warehousing, manufacturing, surveillance or complex video games.

## 7.1 Future Work

However, research on improving M* continues. The first improvement will be to fully implement the Independence Detection framework as described by Standley. This means to implement a tie breaking rule whenever replanning a merged group. Paths that result in fewer collisions with "out-of-group" robots should be prefered to others. This leads to fewer future collisions and thus avoids unnecessary replanning or merging of groups. Early results have shown that this behavior enhances the performance of ID with OD-rM* even more.

The focus of future research lies on combining OD-rM* with another path planning algorithm called Meta-Agent Conflict-based Search (MA-

CBS) [24]. MA-CBS was introduced by Sharon et al. and works by computing individual paths for every robot, avoiding constraints applicable for that robot. If two paths are found to collide, new constraints are added and the paths are updated accordingly. If robots are found to collide more frequently, they are merged into a group and treated as a single meta-agent. The number of conflicts that have to occur until robots are merged is bounded by $B$, a predefined parameter. Independence Detection is a special case of MA-CBS with $B = 0$: Robots are coupled as soon as a collision is detected.

Additionally, future developments will include Enhanced Partial-Expansion A* (EPEA*) [9] as alternative for Operator Decomposition. EPEA* incorporates a priori knowledge of the problem domain (i.e. the relative direction of the goal location to the robot's current position). When expanding a node, only child nodes with optimal cost are put onto the open list. The parent node is again added to the open list, but the cost is revised to reflect the cost of the next best child.

The performance of OD-rM* will be compared to M* using EPEA* instead of OD. Moreover, both variants will be combined with MA-CBS to analyze the performance differences concerning runtime and the number of instances solved.

At the University of Applied Sciences in Salzburg, real-world tests will be conducted to assess the simulation results. It is planned to use 5 to 10 mobile *R-one* robots developed by James McLurkin at Rice University, Houston, TX[2]. The robots will move in an artificial grid environment including obstacles similar to the test instances described in the results section and will be assigned random start and goal positions. Another goal is to automatically determine start and goal positions by using a camera on the ceiling to read in the corresponding positions.

[2]For more details see `http://engineering.rice.edu/NewsContent.aspx?id=3491`

## References


[1] Bellman, R.: *Dynamic Programming.* Rand Corporation Research Studies. Princeton University Press, 1957.

[2] Berg, J. van den and Overmars, M.: *Prioritized Motion Planning for Multiple Robots.* In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2217–2222, 2005.

[3] Berg, J. van den et al.: *Centralized Path Planning for Multiple Robots: Optimal Decoupling into Sequential Plans.* In *Proceedings of Robotics: Science and Systems*, 2009.

[4] Buckley, S.: *Fast Motion Planning for Multiple Moving Robots.* In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pp. 322 –326, May 1989.

[5] Choset, H. et al.: *Principles of Robot Motion: Theory, Algorithms, and Implementation.* MIT Press, 2005 - ISBN 9780262033275.

[6] De Berg, M. et al.: *Computational Geometry.* Springer-Verlag Berlin Heidelberg, 2000 - ISBN 9783540779735.

[7] Edelkamp, S. and Schroedl, S.: *Heuristic Search: Theory and Applications.* Morgan Kaufmann. Elsevier Science, 2011.

[8] Erdmann, M. and Lozano-Perez, T.: *On Multiple Moving Objects.* Algorithmica, 2(1):477–521, 1987 - ISSN 0178-4617.

[9] Felner, A. et al.: *Partial-expansion a* with selective node generation.* Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI), 2012.

[10] Ferner, C., Wagner, G., and Choset, H.: *OD-M*: A New Optimal Algorithm for Multirobot Path Planning.* In *Proceedings of IEEE International Conference on Robotics and Automation*, 2012. Working Title. To be submitted.

[11] Hart, P.E., Nilsson, N.J., and Raphael, B.: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions of Systems Science and Cybernetics, SSC-4(2):100–107, 1968.

[12] Hayes, B.: *Quasirandom Ramblings.* American Scientist, 99(4):282–286, 2011.

[13] Kant, K. and Zucker, S.: *Toward Efficient Trajectory Planning: The Path-velocity Decomposition.* The International Journal of Robotics Research, 5(3):72, 1986 - ISSN 0278-3649.

[14] Kavraki, L.E. et al.: *Probabilistic Roadmaps for Path Planning in High-dimensional Configurations Spaces.* IEEE Transactions on Robotics and Automation, 12:566–580, June 1996.

[15] Korf, R.: *Depth-first Iterative-deepening: An Optimal Admissible Tree Search.* Artificial intelligence, 27(1):97–109, 1985.

[16] Krishna, K.M., Hexmoor, H., and Chellappa, S.: *Reactive Navigation of Multiple Moving Agents by Collaborative Resolution of Conflicts.* Journal of Robotic Systems, pp. 249–269, 2005.

[17] Latombe, J.C.: *Robot Motion Planning.* Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1991 - ISBN 079239206X.

[18] LaValle, S.M. and Kuffner, J.J.: *Randomized Kinodynamic Planning.* In *Proceedings of IEEE International Conference on Robotics and Automation*, 1999.

[19] LaValle, S.: *Planning Algorithms.* Cambridge University Press, 2006.

[20] Liu, J. and Wu, J.: *Multi-Agent Robotic Systems.* CRC Press International Series on Computational Intelligence. CRC Press LLC, 2001 - ISBN 084932288X.

[21] National Instruments Corporation: *An Introduction to A* Path Planning (using LabVIEW).* `https://decibel.ni.com/content/docs/DOC-8983`(03/07/2012), 2011.

[22] Pearl, J.: *Heuristics - Intelligent Search Strategies for Computer Problem Solving.* Artificial Intelligence Series. Addison-Wesley Publishing Company Inc., 1984.

[23] Saha, M. and Isto, P.: *Multi-Robot Motion Planning by Incremental Coordination.* In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5960–5963, Oct. 2006.

[24] Sharon, G. et al.: *Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding.* In *SoCS*, 2012.

[25] Sharon, G. et al.: *The Increasing Cost Tee Search for Optimal Multi-agent Pathfinding.* In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[26] Silver, D.: *Cooperative Pathfinding.* In *Proceedings of the 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[27] Standley, T.: *Finding Optimal Solutions to Cooperative Pathfinding Problems.* In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[28] Standley, T. and Korf, R.: *Complete Algorithms for Cooperative Pathfinding Problems.* In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence.* IJCAI, 2011.

[29] Wagner, G. and Choset, H.: *M *: A Complete Multirobot Path Planning Algorithm with Performance Bounds.* IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3260–3267, 2011.

[30] Wagner, G., Kang, M., and Choset, H.: *Probabilistic Path Planning for Multiple Robots with Subdimensional Expansion.* In *Proceedings of IEEE/RSJ International Conference on Robotics and Automation*, May 2012.