Coupled and Decoupled Approaches to Multi-Robot Motion Planning

Master's Thesis

submitted by

Smith, Mathew James

at the

Universität Konstanz

Chair for Algorithmics

Department of Computer Science

1. Reviewer: Prof. Dr. Sabine Storandt

2. Reviewer: Dr. Georgiana Caltais

Konstanz, 2021

Coupled and Decoupled Approaches to Multi-Robot Motion Planning

Mathew James Smith

Abstract

This thesis explores the role of coupling in the Multi-Robot Motion Planning problem (MRMP). Approaches to MRMP can be broadly categorised as either *coupled* or *decoupled*. It is widely taken to be the case that decoupled approaches, while much more efficient than their coupled counterparts, cannot be complete and optimal. This has however remained a matter of empirical observation. In this work, an algorithm-independent method of measuring the *degree of coupling* of an algorithm is developed. This in turn makes it possible to prove that (almost) complete coupling is necessary to guarantee completeness and optimality in general. Through the way coupling is defined, it follows that the size of the search-space grows exponentially with coupling, and thus the expected trade-off between performance and optimality is made rigorous. *Dynamically coupled* algorithms emerge from this as a compelling candidate to manage this trade-off, and their efficacy is verified through experiment. Dynamic coupling also holds a distinct analogy with the phenomenon of *gradual complexification* exhibited in naturally occurring complex systems. This motivates treating MRMP as an evolving complex system where solutions with the fewest collisions, arrived at with the least coupling, are the fittest, and a genetic algorithm for MRMP is designed based on this concept.

Contents

Preface

Multi-Robot Motion Planning (MRMP) is the problem of, given a set of robots distributed in space, each with a target position, finding a plan which takes each robot from its starting position to its target position in the least number of moves *without collisions*. The constraint of collision avoidance is central to the complexity of the problem: without this constraint, it would be a simple matter of independently finding the shortest paths for each robot—a problem for which polynomial-time algorithms have long been known. Adding the deceptively simple constraint of collision avoidance launches us into the territory of NP-hardness.

MRMP is largely prioritised due to its relevance to valuable practical problems such as automated warehousing, traffic control, network routing, and the control of systems with many degrees of freedom more broadly. The relevance is clear and certainly is reason enough to make MRMP a compelling topic of research. Yet there is something far more fundamental about the problem of MRMP which should not be ignored: many things simultaneously making their way towards some target, all trying (hoping) to do so without conflict. What if the world only had one thing? It would look very different. Granted many things, what if all of these things were never in motion? The world would look very different, in that it would always look the same. What if collisions did not happen, or did not matter? If physical or ideological conflict had no impact on the course of things? The world would look very different. And what if, granted the significance of conflict, resolving it was a consistently easy task? The world would certainly look *very* different. From the motion of physical objects through space, to the constant drive of evolution across species, to the exchange of ideas and resources within a society, everything is in some sense in motion (undergoing change) and everything is in some sense exposed to the threat of collisions: events which impede motion or undermine desired change. In this way, MRMP is an algorithmic abstraction and simplification of a fundamental problem constantly being played out in the world in many forms.

Let's say that a system of many things would like to proceed towards a desired

state without collisions. How should the things in the system organise themselves to achieve this goal? Quickly we can identify two broad categories of approaches. The things could coordinate intensively with one another at every step of the way—every move decided upon by the system as a whole—a *coupled* approach. Or, the things could proceed independently as they see fit, perhaps coordinating with other individuals as necessary—a *decoupled* approach. We will see that there is a clear trade-off between coupled and decoupled approaches. Coupled approaches provide theoretical certainty in finding the best, collision-free plan. But actually finding that plan can easily become insurmountably complex. Decoupled approaches can arrive at a plan quickly, but cannot promise that the plan will be without flaws. It is through this lens—the lens of coupling and its associated trade-offs—that MRMP is analysed in this thesis.

Thesis Contributions

The key contributions of this thesis are summarised as follows:

- An algorithm-independent definition of coupling (Definition 3.1.2), which makes it possible to categorise MRMP algorithms in terms of their *maximum degree of coupling* (Definition 3.1.3), then analyse the behaviour and limitations of all algorithms with a given degree of coupling.
- A proof that in order to guarantee completeness in generality, an MRMP algorithm needs a maximum degree of coupling of at least $n - 1$ (Theorem 4.2.1). Similarly, in order to guarantee optimality in generality, an MRMP algorithm needs a maximum degree of coupling of n (Theorem 4.3.1).
- Identification of the analogy between the phenomenon of *gradual complexification* in complex systems, and *dynamic coupling* employed by a particular class of MRMP algorithms (Chapter 6).
- Definition of a fitness function that allows MRMP to be framed as a complex system (Equation 6.5).
- Design of a Genetic Algorithm for MRMP (ConstraintGA), motivated by the analogy with complex systems (Chapter 8).

Thesis Structure

This thesis is divided into three parts. Part 1 provides background for the later parts. Chapter 1 defines MRMP as it is treated in this work, describes the historical back-

ground and state-of-the-art of the problem, and details algorithms that will be relevant throughout: M*, CBS, and Priority-Planning. Chapter 2 introduces the research questions that will be answered experimentally, details the experimental setup, and summarises the experiments carried out in later chapters. Part 2 focuses on the theory of coupling in MRMP. Chapter 3 defines an algorithm-independent means of measuring coupling. Chapter 4 uses the ideas from the previous chapter to prove limitations on the completeness and optimality of MRMP algorithms based on their degree of coupling. Chapter 5 details experiments that measure how much coupling occurs in dynamically coupled algorithms in practice and its impact on performance, providing empirical backing for the theoretical ideas in the previous chapters. Part 3 is motivated by relating MRMP to complex systems. Chapter 6 introduces complex systems, the phenomenon of gradual complexification, draws an analogy between it and dynamic coupling in MRMP, and defines a fitness function for MRMP based on these ideas. Chapter 7 provides a brief introduction to Genetic Algorithms. Chapter 8 describes a Genetic Algorithm designed for MRMP, bringing together the ideas from the previous chapters, and provides an empirical evaluation of the algorithm. The final chapter concludes the thesis and considers directions for future work.

Part I

Introduction

Chapter 1

Background

1.1 Problem Definition

In this work I focus on multi-robot motion planning on a discrete, two-dimensional, four-connected grid in discrete time with stationary obstacles. Robots are of unit-square size and move at unit-speed in each discrete time-step. Any geometric overlap of robots is considered a collision. There are many variants of the problem beyond that just described. Robots may have varying shapes and sizes. Similarly with obstacles, which themselves may also be in motion. The space and time domains could be continuous rather than discrete. In the continuous case, robots may be able to move at varying speeds, and in any case there may be varying restrictions on how robots are able to move. There may also be differences in what constitutes a collision. The presence of so many degrees of freedom in the definition of the problem poses a challenge when reviewing the literature. Many papers focus on methods for a single variant or a subset of variants, and it is not always clear whether the results and algorithms presented in one paper generalise or are applicable to other variants. Stern et al. provide a survey with definitions of key variants of the problem in [30]. The variant addressed here can reasonably be thought of as a simplified idealisation of the problem which captures its core features. This allows for a focused investigation into the algorithmic properties of the problem, the results of which should be largely transferable to variants with more complicated constraints. With this in mind, a formal definition of the problem as it will be addressed in this thesis follows.

Definition 1.1.1 (Multi-Robot Motion Planning)**.** *An instance of the Multi-Robot Motion Planning problem (MRMP) is defined as a quintuple* $I = (S, T, O)$ *where:*

- $S : \langle \mathbb{Z}^2 \rangle$ *is a sequence of robot starting positions, such that* $S(i)$ *corresponds to the*

starting position of the ith robot.

- $T : \langle \mathbb{Z}^2 \rangle$ *is a sequence of target positions, such that* $T(i)$ *corresponds to the target position of the ith robot.*
- $O : \{\mathbb{Z}^2\}$ *is a set of obstacles: positions onto which robots cannot move.*

The size of I is determined in terms of:

- $n = |S| = |T|$*: the number of robots.*
- $b = (b_{nw}, b_{se})$*: the bounding-box, which specifies the minimal rectangular region containing all positions in S, T, and O. b is defined as a pair of positions: b_{nw} corresponding to the north-west-most position, and b_{se} corresponding to the south-east-most position.*
- m_w *and* m_h*: the width and height of the bounding-box, respectively.*

$P : \langle \langle \mathbb{Z}^2 \rangle \rangle$ *is a plan solving I: a sequence of paths where* $P(i)$ *is a path taking the ith robot from* $S(i)$ *to* $T(i)$*. The path itself is defined as a sequence of positions. A plan is feasible if the paths of all robots can be executed in parallel without collisions. Exactly what constitutes a collision will be defined below.*

Bounded vs. Unbounded

Different algorithms and applications may treat the environment as bounded or unbounded: in the bounded case, robots cannot move outside the bounding box. In the unbounded case, they can, giving robots theoretically infinite space in which they can move. Throughout we will assume the bounded case unless specified.

Moves

The environment is treated as a four-connected grid. Thus, from any given position, a robot has up to five possible choices of moves: **north**, **south**, **east**, **west**, or **remain**. The presence of obstacles, or being at the edge of the bounding box in the bounded case, may render some moves unavailable from certain positions.

Objectives

There are two common objectives for MRMP:

- **Makespan**: Minimise the amount of time until all robots have reached their targets:

$$\min_{P \in S} \{ \max_{i=1 \ldots n} |P(i)| \}$$

- **Total-distance**: Minimise the total distance travelled by all robots:

$$\min_{P \in \mathcal{S}} \{ \sum_{i=1}^{n} |P(i)| \}$$

Where $P \in \mathcal{S}$ denotes a plan P in the solution-space $\mathcal{S}$. When not explicitly specified, it can be assumed throughout that we are working with the makespan objective. Typically, transferring techniques from one objective to another requires nothing more than a change of cost and/or heuristic functions.

Configurations

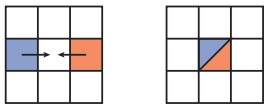
A configuration $C : \langle \mathbb{Z}^2 \rangle$ is a sequence of positions for all robots in the system, such that $C(i)$ is the position of robot i in C. Some of the previous definitions are in fact in terms of configurations: S and T are themselves configurations. In addition to a plan (a sequence of paths for each robot), a solution to an MRMP instance may also be framed as a sequence of configurations. The *configuration-space* of an instance consists of all permutations of robot positions. The configuration-space can be usefully structured as a graph where given configurations C and C', an edge between C and C' exists if the position of every robot in C is adjacent to its corresponding position in C'. Positions p_i and p'_i are adjacent if p'_i is one of the cells connected to p_i excluding those containing obstacles, or if $p_i = p'_i$. The task of minimising the makespan then becomes that of finding the shortest collision-free path in the configuration-space from S to T. Minimising the total-distance can be achieved by weighting the edges by the number of robots which make a move in the transition between the configurations either side of the edge, then finding the shortest weighted path from S to T. With this graph structure in mind, the type of a configuration-space $\mathcal{C}$ can be defined as $\mathcal{C} : C \longrightarrow \{C\}$, such that $\mathcal{C}(C)$ yields the set of configurations adjacent to C.

Collisions

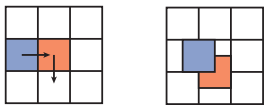
A collision occurs when the trajectories of two or more robots overlap, taking the robots to be equal to the size of a grid cell. Collisions can be placed into two categories (visualised in Figure 1.1):

- **Clash**: occurs when two robots move into the same cell at the same time.
- **Overlap**: occurs when robot a moves onto a cell occupied by robot b, but robot b is moving in a different direction to robot a.

It is important to make clear the relationship between the configuration-space and collisions. Any configuration where two robots are on the same cell causes a clash,

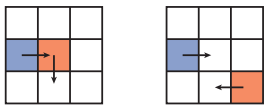


(a) Clash



(b) Overlap

Figure 1.1: The two types of collision.



(a) Two different starting configurations.

(b) Equivalent target configuration.

Figure 1.2: Illustration of how whether or not there is a collision at a configuration depends on which configuration precedes it. While the moves specified from both the left and right configurations in (a) lead to the same configuration shown in (b), the moves from the left configuration result in an overlap, while the moves from the right one are collision-free.

so will lead to a collision regardless of which configuration precedes it. Whether or not moving to a specific configuration causes an overlap, however, depends on which configuration is being moved from. Figure 1.2 illustrates this. Because of this, collisions are associated with edges rather than vertices in the configuration-space.

1.2 Configuration-Space Diagrams

Frequently throughout this work it will be useful to visually represent (subsets of) configuration-spaces. As mentioned above, the configuration-space can be usefully structured as a graph. Plainly visualising an entire configuration-space like any other graph, however, is not useful: we end up with a difficult-to-follow web of edges, often between configurations which aren't relevant to the discussion at hand. Typically, a visualisation will focus on the adjacencies of a specific configuration. This priority can be used to create a more intuitive and useful visualisation. Configuration-space diagrams are structured as *rows* of configurations, where the first row (call it row zero) has just one configuration: the configuration we are focusing on. In most cases this will be the starting configuration of an instance. Row one contains the *first-order ad-*

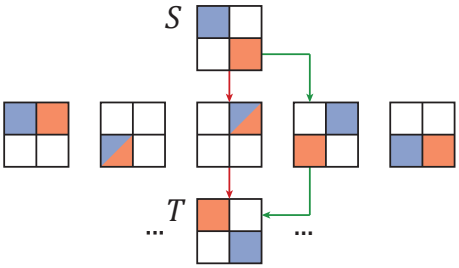


Figure 1.3: Example of a configuration-space diagram. Row zero contains the focused configuration, which is the starting configuration of the instance in question as indicated by the label S. Row one contains all of the first-order adjacencies of S. Row two contains the target configuration T—a second-degree adjacency of S—but it does not contain all second-degree adjacencies, as indicated by the horizontal dots. The green edges highlight a valid, collision-free path from S to T, while the red edges highlight a colliding path.

jacencies of the configuration in row zero, row two the second-order adjacencies, and so on. By default, edges are not drawn between configurations, as they are implied by which row each configuration is in (the relevant edges, at least). Some specific edges may be drawn to highlight particular paths through the configuration-space. Horizontal dots in a row indicate that not all adjacencies of that row have been drawn, which may be necessary because, as will be seen below, the number of adjacencies a configuration can have grows exponentially with the number of robots. The configurations themselves are represented as grids, where each unique colour in the grid represents the position of a unique robot. When multiple robots occupy one cell, the colour of that cell is divided between the colours of each of the robots occupying it. Configurations may have a label at their top left corner, to, for example, indicate the start and target configurations. Figure 1.3 provides an example of a configuration-space diagram.

1.3 Historical Background

The origins of multi-robot motion planning can be traced back to the *15-puzzle* of the late 1800s. In a 15-puzzle, 15 squares are positioned in some permutation on a 4×4-grid with each square having a specific target position. The squares are moved by sliding them into the one free space on the grid. Figure 1.4 provides an illustration of a 15-puzzle. The puzzle gained popularity due to its difficulty, which would later be verified mathematically. Johnson and Story in 1879 [15] showed that only half of the possible starting permutations are solvable using a parity argument: the parity of the permutation of the 16 squares plus the parity of the Manhattan distance of the empty square from the bottom right corner can be used to partition permutations into classes

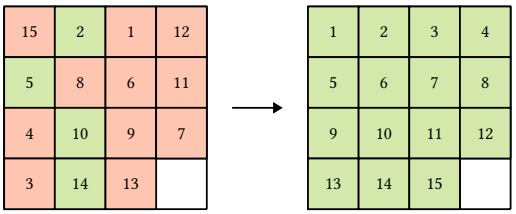


Figure 1.4: An initial permutation of a 15-puzzle (left) and the solved puzzle (right).

of reachable and unreachable states. Ratner and Warmuth in 1990 [22] showed that finding the smallest sequence of moves to solve an instance of the problem generalised to $n \times n$-grids (i.e. $n^2 - 1$-puzzles) is NP-Hard. Due to their inherent difficulty, a valuable use for such puzzles has been benchmarking heuristic functions for search techniques such as A*.

Contemporary work on the problem has evolved alongside the field of computational geometry, as well as with the increasing significance of swarm robotics, and other domains necessitating the coordination of many objects moving through space, such as automated warehousing and traffic control. An early result on the algorithmic side was the discovery of algorithms due to Ramanathan and Alagar [21] and Schwartz and Sharir [27] for coordinating the motion of several disks among polygonal obstacles, with runtimes polynomial in the number of obstacles, but exponential in the number of disks. Hopcroft, Schwartz, and Sharir [13] showed that deciding whether a given target configuration is reachable in such a setup is PSPACE-complete. Abellanas et al. [1] and Bereg, Dumitrescu, and Pach [2] contributed to the challenge of minimising the number of moves of disks moving into a target configuration, developing algorithms that provide upper and lower bounds on the number of moves. Dumitrescu and Jiang [9] improved on these bounds, and showed that the problem is NP-hard, even for disks of homogeneous size and shape.

On the practical side, recent success has been seen in applying algorithms for variants of MRMP to automated warehousing [32], swarm robotics [5, 24], and traffic control [11].

1.4 Complexity

Immediately, we can see that the size of the configuration-space and the graph that it induces is exponential in the number of robots, providing initial evidence for the hardness of MRMP: from any given configuration C, each of the n robots can choose from up to five possible moves (one of the four directions, or to remain in place), so the number of configurations adjacent to C is bounded above by 5^n. For a $m_1 \times m_2$

sized bounding-box, an upper-bound of $\mathcal{O}(m_w + m_h)$ can be placed on the makespan, by virtue of approximation algorithms from Demaine et al. and Yu [7, 34]. More specifically, it is guaranteed that the makespan will be within a constant factor of the maximum distance of any one robot to its target, known as a *constant stretch factor*. So, while the length of the path from S to T is independent of n and linear in the size of the bounding box, because each vertex along that path could have a number of adjacencies exponential in n, ditto for higher-order adjacencies, the size of the configuration-space explodes with n, making a typical path-finding approach through the configuration-space intractable beyond small instances. Demaine et al. [7] show that minimising the makespan is NP-hard by demonstrating a polynomial-time reduction from Monotone 3-Sat.

1.5 State-of-the-Art

Approaches to MRMP can be broadly categorised into *coupled* and *decoupled* approaches. Coupled approaches plan the movements of robots collectively, typically by searching the configuration space. While being able to guarantee optimal solutions, coupled approaches are only able to solve relatively small instances due to the exponential size of the configuration-space. SAT-solvers and ILP formulations [16, 14, 35] have been used to make the most of the coupled approach, and sampling methods [14, 25, 29] have been used to improve performance at the cost of possibly returning a suboptimal solution. Decoupled approaches plan the paths of each robot individually, then eliminate collisions through a minimal amount of inter-robot communication. Decoupled approaches are typically much faster than their coupled counterparts, with the trade-off that no such approach has been developed that is guaranteed to produce an optimal solution. Furthermore, decoupled approaches are often vulnerable to incompleteness: failing to return even a feasible solution when one exists [31].

A common decoupled approach is *Priority-Planning* [8, 23, 3]. First, each robot is assigned a priority according to some heuristic (for example, by distance to the target). Then, the shortest path is found for the highest priority robot and added to the plan. The path for the next highest priority robot is then found by finding the shortest path possible while treating the previous robot as a moving obstacle along its shortest path. This process repeats until all robots have been planned for, each treating the robots before them as moving obstacles. The quality of the solution returned by a priority-planning based method, and if it returns one at all, is highly sensitive to the choice of heuristic used to assign priorities.

Two-phase decoupled approaches are also popular. In the first phase, a feasible so-

lution is computed without concern for optimality. Priority-planning using a heuristic that is likely to return a feasible solution, but not necessarily an optimal one, is typical for the first phase. In the second phase, the initial solution is improved upon using local search methods. There is much variation in the local search methods used. The three winning teams of the 2021 CG:SHOP Challenge [10], in which MRMP was the problem of choice, all used a two-phase approach with different local search methods. Crombez et al. [6] iteratively improved the paths of robots in the initial solution, in some cases allowing the feasibility of the solution to be "broken", before arriving at a new feasible solution with improved makespan. Liu et al. [17] used at k-opt approach, in which the paths of k sampled robots were iteratively improved, considering a number of heuristics for assigning sampling probabilities. Yang and Vigneron [33] used simulated annealing to guide a combination of stretching and tightening actions on the paths of individual robots in order to improve the initial solution.

In between coupled and decoupled approaches lie *dynamically coupled* approaches. Such approaches begin in a decoupled fashion, then increase the dimensionality of the search-space as collisions are encountered. Typically, such approaches will become fully coupled in the worst-case, but in practice may find an optimal solution while maintaining a low-dimensional search-space thus having faster average run-times. Two such dynamically coupled algorithms: M* and CBS, will be introduced in detail below, and will be focal points throughout this thesis work. M* is of particular interest as it makes explicit the role of coupling in MRMP, and gestures towards a means of quantifying coupling. CBS takes more of an implicit, fine-grained approach to increasing coupling, which we will see tends to result in a smaller subset of the configuration-spaced being searched, and therefore faster practical performance, compared to M*.

1.5.1 M* and Sub-dimensional Expansion

M*, proposed by Wagner and Choset [31], is a variant of A* search specifically designed for MMRP. The main difference between A* search on the configuration-space and M* is the use of *sub-dimensional-expansion*. M* begins by assuming that all robots can take the shortest path from their start locations to their targets without collisions. Thus it begins searching a sub-graph of the configuration-space that is a path-graph from S to T, and can be considered a one-dimensional search-space. The algorithm then traverses the search-space as in A*. When a collision is encountered, the dimensionality of the search-space is *expanded* by including the configurations induced by considering all possible moves of robots that collided, rather than just the moves on their shortest paths. This process continues until an optimal (relative to a provided

heuristic function) collision-free path from S to T is found.

Pseudocode for M* is provided in Algorithm 1. The **shortestpaths** subroutine initialises the search-space as the sequence of configurations resulting from each robot taking its shortest path. The queue is a priority queue prioritising vertices in ascending order by v.cost $+h(v)$, where h is the provided heuristic. The **backtrack**(v) subroutine reconstructs the optimal path by backtracking through the parents, starting from v. The **expand(**v**)** subroutine performs sub-dimensional expansion—adding the adjacent configurations to v resulting from allowing robots in the collision-set of v to consider all possible moves, while robots not in the collision-set only take their optimal move. The **backprop**(v, w.collset, queue) subroutine propagates the collision-set of w through all vertices in paths leading to w, and adds those vertices back into the queue if their collision-sets changed. The **collisions**(v, w) subroutine finds collisions that occur during a transition from v to w.

Algorithm 1 M*

```
function M*(I = (S, T, O))
    V ← shortestpaths(I)
    for v ∈ V do
        v.cost ← ∞
        v.backset ← ∅
        v.collset ← ∅
    end for
    S.cost ← 0
    queue.enqueue(S)
    while queue is not empty do
        v ← queue.dequeue()
        if v = S then
            return backtrack(v)
        end if
        for w in expand(v) do
            w.backset.append(v)
            w.collset ← w.collset ∪ collisions(v, w)
            backprop(v, w.collset, queue)
            if collisions(v, w) is empty and v.cost + cost(v, w) < w.cost then
                w.cost ← v.cost + cost(v, w)
                w.parent ← v
                queue.enqueue(w)
            end if
        end for
    end while
    return no solution found
end function
```

The completeness and optimality of M* follows from the already established results for A*, combined with the fact that M* can be interpreted as alternating between

searching the already expanded search-space using A*, and further expanding the search-space based on the partial search results. M* expands the search-space each time the collision set of a vertex is updated. The collision-set of a vertex can be updated at most $n - 1$ times (once for each robot, but the first update must add at least two robots). Therefore, the search-space can be expanded at most $|V|(n-1)$ times, and so after a finite number of steps the A* planner will have either found an optimal path, or exhausted the entire configuration-space. Full details of the proofs of optimality and completeness are provided in section 4.3 of [31].

As with A*, completeness and optimality are dependent on the provided heuristic being *admissible*. A heuristic is admissible if it is optimistic, i.e. it never overestimates the distance of a node from the destination. In M*, the input to a heuristic function must be a configuration. For minimising the makespan, an admissible heuristic is the *maximum Manhattan distance* of any one robot in the configuration to its target. For minimising the total-distance, an admissible heuristic is the *sum of induced costs*: the sum of the Manhattan distances of all robots in the configuration to their targets. M* can be turned into an approximation algorithm by *inflating* the heuristic, that is, multiplying by some $\epsilon > 1$. At the cost of optimality, doing so improves performance by causing the algorithm to favour expanding nodes which bring it closer to a solution. Inflation by ϵ maintains an ϵ-approximation factor.

1.5.2 Conflict-Based Search

Conflict Based Search (CBS) is another complete and optimal search algorithm for MRMP, proposed by Sharon et al. [28]. CBS does not explicitly search the joint configuration-space. Rather, individual robot paths are gradually updated as we build and traverse a *constraint tree*. Each vertex v in the constraint tree consists of:

- **Constraints** (v.constraints). A constraint is a tuple (r, p, t), which enforces that robot r cannot move onto position p at time t.
- **Plan** (v.plan). A sequence of paths taking each robot from start to target as previously defined.
- **Cost** (v.cost). The cost of v.plan, according to the chosen objective.

The search begins by creating the root node, consisting of an empty constraint set, and a plan and cost yielded by each robot taking its shortest path from start to target, ignoring other robots. Then, collisions are detected in the initial plan. If there are no collisions, the initial plan is returned. Otherwise, one of the collisions are chosen. A collision is represented as a tuple (R, p, t) where R is the set of colliding robots, p is

the position and t the time at which the collision occurred. For each $r \in R$ a child vertex is created by adding the constraint (r, p, t) to the constraint set of the root. The plans and costs of the child nodes are arrived at by updating them to obey the newly added constraints. The constraint tree is in this way built and traversed in a best-first fashion according to cost until a collision-free plan is found. Algorithm 2 provides pseudocode for CBS. Note that the subroutine **shortestpath**(r_i, w.constraints) computes the shortest path for robot r_i from start to target, while obeyingthe providede constraint set. This constitutes what is referred to as the "low-level search" in the original paper by Sharon et al., and can be implemented based on a single-robot search algorithm such as A*.

Algorithm 2 Conflict-Based Search

```
function CBS(I)
    root.constraints = ∅
    root.plan ← shortestpaths(I)
    root.cost ← makespan(root.plan)
    queue.enqueue(root)
    while queue is not empty do
        v ← queue.dequeue()
        if v.solution is collision-free then
            return v.solution
        end if
        c ← (R, p, t), the first conflict in v.solution
        for r_i ∈ R do
            w.constraints ← v.constraints ∪{(r_i, p, t)}
            w.plan ← shortestpath(r_i, w.constraints)
            w.cost ← makespan(w.plan)
            if w.cost < ∞ (a plan was found) then
                queue.enqueue(w)
            end if
        end for
    end while
end function
```

For simplicity, the above only accounts for *clash* collisions, disregarding overlaps. It is straightforward to modify CBS to account for overlaps by distinguishing *vertex constraints*, of the form (r, p, t) as already seen, and *edge constraints*, of the form (r, p_{src}, p_{dst}, t), which enforces that robot r cannot move onto position p_{dst} *from* position p_{src} at time t.

The optimality of CBS follows from the best-first traversal of the constraint tree. Given two costs $x < y$, CBS will expand all vertices of cost x before expanding any of cost y. Thus when a collision-free plan is found, it is guaranteed that only the minimal necessary amount of re-routing, compared to the initial solution, has occurred. CBS

can be shown to always return a solution when one exists by showing that for any given cost, there is a finite number of possible vertices with that cost. CBS is however not complete on its own, as it may fail to recognise when a solution does not exist: it can create constraints for an infinite number of time-steps. CBS can be made complete by supplementing it with a linear-time feasibility checking algorithm, as presented by Yu and Rus [36]. Full details of the proof of completeness and optimality for CBS are presented in Section 5 of [28].

While the low-level search performed in the creation of each vertex in the constraint tree is a single-robot one and thus achievable in polynomial-time, the number of vertices in the tree may grow exponentially with respect to n, and thus the worst-case run-time of CBS is exponential. In practice, however, the constraint tree often does not need to grow very large. Consequently, CBS has been shown to have significantly improved performance compared to methods that search the joint configuration-space.

Chapter 2

Experiments

This chapter provides an overview of the experiments run throughout this work. First, the questions that the experiments seek to answer are provided. Then the experimental setup is detailed, in particular how instances were generated. The final section provides a summary of the experiments run. A full account of each experiment is found in the chapter to which it is relevant, which will be cross-referenced here.

2.1 Questions

Experiments have been run in an effort to answer the following questions:

RQ1. Coupling

1. How much coupling actually occurs when running dynamic fully coupled algorithms?
2. What is the relationship between actual coupling and run-time?
3. How does the run-time of dynamic fully coupled algorithms compare to that of static fully coupled algorithms?
4. How does the occurrence of collisions impact actual coupling and run-time?

RQ2. Genetic Algorithm

1. With a fitness function designed such that low-collision solutions are the fittest, how effective is a genetic algorithm in resolving collisions?
2. How does the solution-quality of the ConstraintGA starting in a simple state then gradually complexifying compare to when it begins in a complex state?

3. Is the performance of dynamic-fully-coupled algorithms improved by starting with initial solutions with fewer collisions, provided by pre-processing with the ConstraintGA?

2.2 Setup

2.2.1 Instances

The instances used for experiments have been generated randomly according to three parameters: n, the number of robots; m, the size of the grid (only square grids have been used, so m indicates an $m \times m$-grid); and θ, the *cluster-factor*, interpreted as follows. At the start of the generation of each instance, $\lceil \frac{n}{5} \rceil$ "cluster points" are chosen in the grid uniformly at random. If $\theta = 0$, cluster-points are ignored, and the instance is created by choosing start and target positions for each robot uniformly at random. If $\theta = 1$, the probability distribution over positions in the grid is weighted such that those closest to a cluster-point have the highest probability. In general, the weighting of a position p is determined in terms of θ accordingly:

$$\frac{1-\theta}{n} + \theta \frac{(d_{\max} - d_p)^2}{(\sum_{q \in G} d_q)^2} \tag{2.1}$$

Where d_p is the shortest distance of position p from one of the cluster-points, $d_{\max}$ is the largest possible distance between two points in the grid (i.e. $m_w + m_h$), and G is the set of positions in the grid. It can be seen that when $\theta = 0$, the weight is entirely determined by the left term, resulting in a uniform weighting, while if $\theta = 1$, the weight is entirely determined by the right term, resulting in a heavy weighting towards the cluster-points. θ's between 0 and 1 result in a "mix" of the two terms.

Creating instances with varying cluster-factors ensures that the generated instances embody a diverse range of scenarios. In particular, it is expected that highly clustered instances will have higher incidences of collisions. Having increasing incidences of collisions not only as n increases but within the instances for each value of n will allow for a more confident analysis of RQ1.4. Figure 2.1 illustrates instances generated with varying values of θ. Figure 2.2 illustrates measurements showing that higher clustering leads to more collisions.

All experiments used instances on 8×8-grid's, with n ranging from 4 to 16. For every value of n, 20 instances were generated: five for each of $\theta = 0, 0.33, 0.66, 1$.

Figure 2.1: Starting configurations of randomly generated instances with $n = 20$, $m = 16$, and $\theta = 0, 0.33, 0.66$, and 1 respectively from left to right.

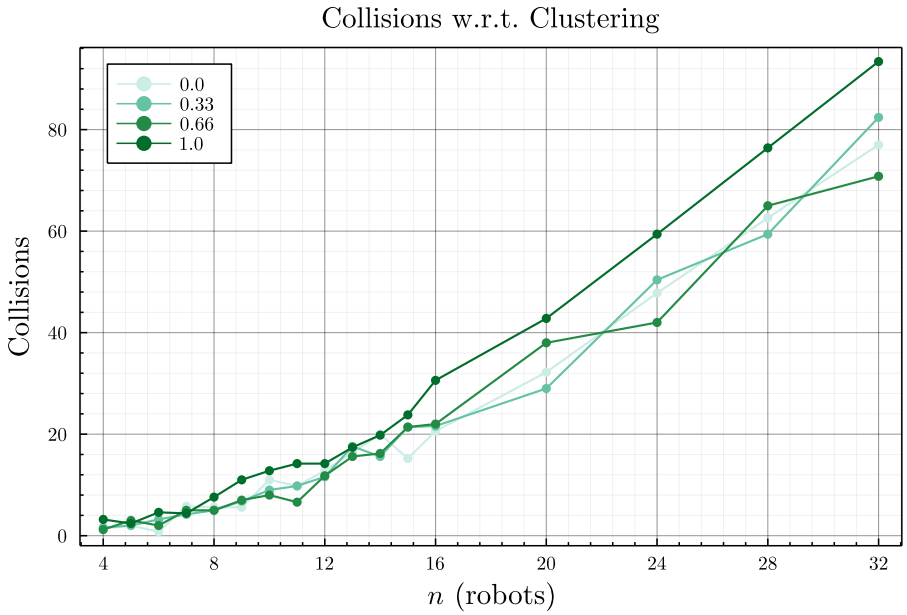


Figure 2.2: The number of collisions that occur when routing each robot according to its shortest path, grouped by the cluster-factor. It can be seen that $\theta = 1$ consistently results in more collisions.

2.2.2 Hardware

Experiments were run on an AMD Ryzen Threadripper 3970X CPU with 32 cores, 64 threads, running at 3.7GHz, and with access to 256GB of RAM. Parallelism was not used in most algorithms tested, with the exception of the ConstraintGA.

2.3 Summary of Experiments

2.3.1 Empirical Analysis of Coupling

To answer the question set RQ1, CBS, M*, and A* were run on 8 × 8-grid instances with n ranging from 4 to 16. Run-times and coupling were measured. Additionally, the number of collisions occurring in "naive solutions", that is routing each robot according to its shortest path, completely disregarding other robots, was measured. The experiments showed a significant increase in performance of the dynamically coupled algorithms (CBS and M*) compared to A*. The "actual coupling" occurring in CBS was on average much lower than M*, which translated into significantly improved run-times. Finally, there was a strong correlation between the number of collisions in the naive solutions, actual coupling, and run-time. The results are presented in full and interpreted in context in Chapter 5.

2.3.2 Genetic Algorithm

To answer the question set RQ2, the ConstraintGA was run on 8 × 8-grid instances with n ranging from 4 to 16. Both solution quality (in terms of collisions), and coupling (in terms of active constraints), were measured. On a few individual instances, fine-grained behaviour of the ConstraintGA exhibiting gradual complexification as designed was compared to a variation where it did not. The experiments showed that in most instances, the ConstraintGA was able to resolve more than 50% of collisions. They also showed that gradually complexifying in terms of the number of active constraints was beneficial, making the system more evolvable. Providing the output of the ConstraintGA to CBS as a preprocessed initial solution provided little benefit. The results are presented in full and interpreted in context in Chapter 8.

Part II

The Theory of Coupling

Chapter 3

Quantifying Coupling

The goal of this chapter is to make the notion of coupling in MRMP precise. With an algorithm-independent means of quantifying coupling in hand, in later chapters it will be possible to formally analyse the role that it plays in the limitations and complexities of MRMP algorithms.

3.1 Definition

Coupling is a widely used concept in the MRMP literature. Yet there isn't a mathematically rigorous definition of it. Such a definition will be necessary in order to formally analyse the role that it plays in the capabilities of MRMP algorithms. Any proofs and claims made based on that definition can only be as strong as the definition itself. Much thought and care, then, needs to go into ensuring that this definition convincingly and compellingly captures what we mean when we talk about coupling in MRMP.

Intuitively, a robot is coupled if it abdicates responsibility for its decision-making to the greater system. At each point in time, the system can consider the possible next moves of the coupled robot, and with that in combination with the other information available to it (i.e. the current positions and targets of all robots, plus the possible next moves of other coupled robots) make a decision on how the coupled robot should proceed. When a robot is not coupled, only its actual position at any point in time is known to the greater system. From the system's perspective, the decoupled robot autonomously decides where to go next. The system has no say in—only observes—the decision-making of the decoupled robot.

Algorithmically, this coupling can be realised as an increase in the dimensionality of the (available) configuration-space. In a one-dimensional configuration-space, each

robot follows a fixed path. It is simply a path graph from S to T (assuming such a path is possible). We can only hope that the path does not involve any collisions. A two-dimensional configuration-space means that one robot has become coupled—at each step away from S, all possible moves that the coupled robot could make in the next step away from S are represented in the configuration-space, while the remaining uncoupled robots continue to take their one fixed move. The coupled robot provides *information* and *choice* to the system, manifested as additional adjacencies at each step away from S, which could potentially allow it to avoid collisions. Multiple robots could be coupled, further increasing the dimensionality of the configuration-space, and hence the information and choice available to the system. If all robots are coupled, the resulting n-dimensional configuration-space is total.

One might argue that a dimensionality-based definition of coupling is not general enough, because not all algorithms are searches of the configuration-space. For example, we have seen that CBS searches a fundamentally different space: the constraint tree, in order to arrive at a solution. I argue that all algorithms at-least involve an *implicit* search of the configuration-space: there is some subset of the configuration-space that the algorithm *observes* throughout its execution. This will be termed the *Observed Configuration-Space*:

Definition 3.1.1 (Observed Configuration-Space)**.** *The observed configuration-space of an algorithm $\mathcal{A}$ running on an MRMP instance I, denoted $\mathcal{B}_{\mathcal{A}}(I)$, is the subset of the total configuration-space that appears in the memory of the machine running $\mathcal{A}$ on the input I.*

When $\mathcal{A}$ and I are unbound, simply $\mathcal{B}$ will be used to refer to the notion of observed configuration-space. By reasoning about $\mathcal{B}$, we can reason about the search of the configuration-space that an algorithm performs, be it implicit or explicit. Note that when we say a subset "appears in memory", we mean the cumulative subset that appears over the entire run of the algorithm. The entire subset does not need to be in memory all at one time. Imagine someone, or some machine, is auditing the execution of the algorithm—watching what is in memory at each step. Whenever a configuration from the total configuration-space appears in memory, the auditor adds it to the observed configuration-space. The configurations that the auditor has noted at the algorithm's completion constitute the final observed configuration-space.

The above account of coupling, and the existence of dynamically coupled algorithms, indicate that coupling is not a binary classification between coupled and decoupled. Rather, it exists on a spectrum. Therefore, we will characterise algorithms by their *degree of coupling*, parameterised by n. The degree of coupling of an algorithm $\mathcal{A}$ running on an instance I is determined by $\mathcal{B}(\mathcal{A}, I)$.

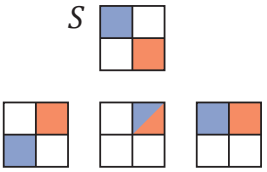


Figure 3.1: Configuration-space diagram illustrating a robot having "multiple moves represented" in the adjacencies of a configuration: the blue robot has three moves represented in the adjacencies of S, while the orange robot only as one.

Definition 3.1.2 (Degree of Coupling)**.** *The degree of coupling of an algorithm $\mathcal{A}$ running on an MRMP instance I, denoted $\kappa_{\mathcal{A}}(I)$, is the dimensionality of $\mathcal{B}_{\mathcal{A}}(I)$. This is equivalent to the number of robots which have* multiple moves *represented in at least one set of adjacencies of a configuration in $\mathcal{B}_{\mathcal{A}}(I)$.*

The above definition indicates that determining $\kappa_{\mathcal{A}}(I)$ boils down to determining $\mathcal{B}_{\mathcal{A}}(I)$, then counting how many robots have multiple moves represented in at-least one set of adjacencies. Figure 3.1 illustrates a robot having multiple moves represented. $\mathcal{B}_{\mathcal{A}}(I)$ could in turn be determined by the auditing method discussed above. Recall that the end goal of this discussion however is to be able to determine a *general* degree of coupling for an algorithm across all instances, then reason about the limitations of *all* algorithms with a given degree of coupling. To this end we will define the *maximum degree of coupling*, $\bar{\kappa}$, as follows:

Definition 3.1.3 (Maximum Degree of Coupling)**.** *The maximum degree of coupling of an algorithm $\mathcal{A}$ on instances of n robots, denoted $\bar{\kappa}_{\mathcal{A}}(n)$, is the highest possible degree of coupling during a run of $\mathcal{A}$ on any instance of n robots.*

Typically, n will remain uninstantiated, and we will refer to the maximum degree of coupling of an algorithm $\mathcal{A}$, parameterised by n, as $\bar{\kappa}_{\mathcal{A}}$. When $\mathcal{A}$ is unbound, just $\bar{\kappa}$ will be used as a synonym for "maximum degree of coupling". "k-$\bar{\kappa}$-algorithms" will be used to refer to all algorithms with maximum degree of coupling equal to k. Exactly how $\bar{\kappa}$ is determined will vary between algorithms. In general, it will involve recognising the largest possible configuration-space that could be observed. In M* for example, the worst-case occurs when all robots encounter a collision at some point: all possible moves for all robots are considered, the observed configuration-space is equal to the total configuration-space, and therefore $\bar{\kappa} = n$.

3.2 Categorising MRMP Algorithms

Based on κ and $\bar{\kappa}$, we can define broad categories for MRMP algorithms. An algorithm $\mathcal{A}$ is:

- **Decoupled** if $\bar{\kappa}_{\mathcal{A}} = 1$.
- **Partially Coupled** if $1 < \bar{\kappa}_{\mathcal{A}} < n$.
- **Fully Coupled** if $\bar{\kappa}_{\mathcal{A}} = n$.
- **Statically Coupled** if $\forall I \in \mathcal{I} : \kappa(\mathcal{A}, I) = \bar{\kappa}_{\mathcal{A}}$.
- **Dynamically Coupled** if $\exists I \in \mathcal{I} : \kappa(\mathcal{A}, I) < \bar{\kappa}_{\mathcal{A}}$.

Where $\mathcal{I}$ is the space of MRMP instances. Notice that any algorithm must be both exactly one of static or dynamic, and exactly one of decoupled, partially coupled, or coupled.

3.3 Analysis of Selected Algorithms

M*

As the definition of coupling is closely related to the notion of sub-dimensional expansion used in M*, establishing $\bar{\kappa}$ for M* is straightforward. If all robots encounter a collision, sub-dimensional expansion expands the configuration-space to consider all possible moves of all robots: all robots are coupled, thus $\bar{\kappa} = n$. Of course, it is also possible that the initial 1-dimensional path is collision free. In such instances, $\kappa = 0$. Therefore, M* is a dynamic-fully-coupled algorithm.

CBS

To derive $\bar{\kappa}$ for CBS, we will step through how it behaves in a pathological scenario where four robots are clustered around the outside of a 3×3-grid such that all want to move through the centre to take the shortest path to their targets. This instance will be labelled I_{C4}. The start (S) and target (T) configurations of I_{C4} are illustrated in Figure 3.3. The partial observed configuration spaces (a) to (e) in this figure illustrate the growth of the observed configuration-space as we traverse the constraint tree. Figure 3.2 shows the constraint tree that is built throughout the search, with the highlighted vertices indicating those that are actually expanded, and the overall path taken by the search. The search begins at the root node of the constraint tree, with no constraints: all robots take their shortest path to target, disregarding others. As a result, all robots collide in the centre, illustrated in (a). Four vertices are added as children to the root: one with a constraint for each robot. All resulting solutions of these vertices have a cost of 5—we will suppose that the left-most vertex is expanded, with a constraint for r_1, represented in blue in the figure. Illustrated in (b), r_1 is rerouted around the

centre in order to obey the constraint, while the other three robots still collide in the centre. This process is repeated for robots r_2 (orange) and r_3 (green), illustrated in (c) and (d). It can be seen in (d) that there remains just one collision: between r_1 and r_4 (purple) at $t = 3$ (i.e. the second-order adjacencies in the configuration-space diagram). A constraint is added for both r_1 and r_4, but the constraint for r_1 results in a makespan of 6 (it would move around r_4), while the constraint for r_4 maintains a makespan of 5, so the latter will be expanded. Obeying this along with all of the previous constraints yields a collision-free path, as illustrated in (e). (e) represents the final observed configuration-space for this run. Inspecting this, we can see that all four robots have multiple moves represented in at least one row of adjacencies, therefore $\kappa_{CBS}(I_{C4}) = 4$.

The above scenario illustrates that in CBS, if all robots are involved in collisions, they may (potentially) all become coupled, and so $\bar{\kappa}_{CBS} = n$. Of course, less coupling is possible. In the best-case scenario, the initial constraint-free solution will have no collisions, and no coupling will be necessary. Therefore, CBS is a dynamic-fully-coupled algorithm.

Notice that CBS is "tighter" than M*: it can find a feasible solution with much less "actual coupling". Imagine a scenario where n robots are paired such that each pair collides once, then there are no further interactions. CBS would add one constraint per pair of robots. Despite the fact that every robot encounters a collision, κ_{CBS} for this instance would be $\frac{n}{2}$. Furthermore, in the observed configuration-space, we would see that the coupled robots only consider alternative moves across a small number of adjacencies where the collisions occurred. In contrast, κ_{M^*} in this scenario would be n, and once each robot is coupled, it will have alternative moves considered at every adjacency thereon.

Priority-Planning

From a coupling perspective, solving an MRMP instance using priority-planning should be viewed as iteratively solving n sub-problems, each itself an MRMP instance. A 1-$\bar{\kappa}$-algorithm is used for each sub-problem. In the nth and final sub-problem, the robot with the lowest priority is the coupled robot, and the "arbitrary choices" of paths for the decoupled robots by the algorithm solving this final sub-problem are determined by the algorithms that solved the previous sub-problems. Therefore, priority-planning is a static-decoupled algorithm.

Why is Priority-Planning considered as sub-problems and CBS not, given that CBS is also made up of single-robot searches at the low-level? Because in CBS, through the constraints, there is implicit awareness of the greater system throughout the ex-

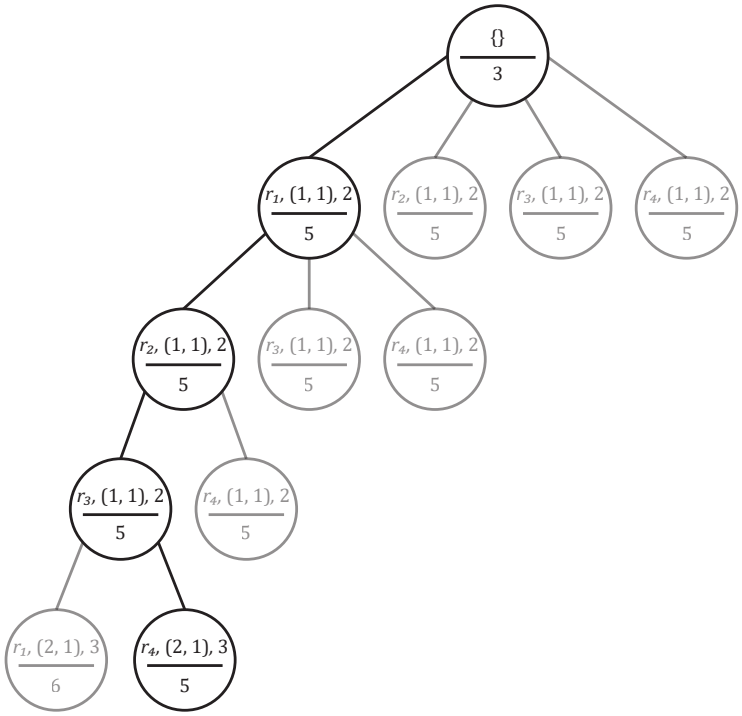


Figure 3.2: The constraint tree resulting from running CBS on the described pathological four-robot instance. The vertex labels are of the form $\frac{\text{constraint}}{\text{cost}}$. The entire constraint set of a vertex is recovered by collecting all constraints on the path from the vertex to the root.

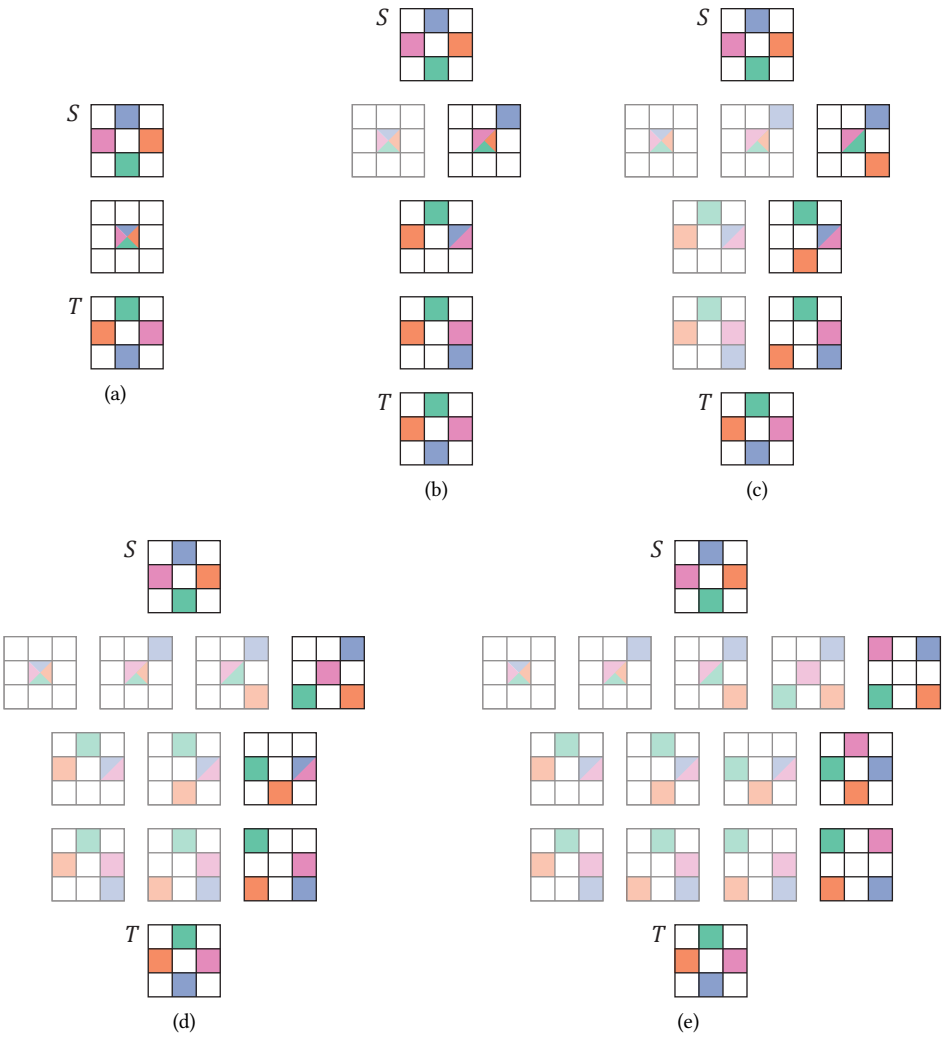


Figure 3.3: Growth of the observed configuration-space for CBS running on the described pathological four-robot instance. For each partial configuration-space, the highlighted configurations in each row together form the plan for the vertex in the constraint tree that was last expanded.

ecution of the algorithm. In Priority-Planning on the other hand, the first robot has no awareness of the lower-priority robots—they essentially do not exist at the time that the plan for the first robot is made, and so configurations observed at this stage cannot be considered a part of that of the overall MRMP instance that the algorithm is solving. Similarly for all other robots up until the last—at this point, the low-level search has an awareness of all robots in the system, and the configurations observed here indeed make up the observed configuration-space.

Chapter 4

Limits

Knowing $\bar{\kappa}$ for an algorithm allows us to place limits on the configuration-spaces that algorithm could possibly observe. A set of "observable configuration-spaces" can in principle be derived for any given instance. Then, the algorithm in question running on the given instance must observe one of (or a subset of one of) those observable configuration-spaces. Reasoning about these observable configurations-spaces in turn allows us to place certain limitations on the capabilities of MRMP algorithms purely based on $\bar{\kappa}$. In this chapter it is shown that $\bar{\kappa} \geq n - 1$ is necessary to guarantee completeness, and $\bar{\kappa} = n$ is necessary to guarantee optimality.

4.1 Observable Configuration-Spaces

Definition 4.1.1 (Observable Configuration-Spaces)**.** *The observable configuration-spaces for a k-$\bar{\kappa}$-algorithm running on an instance I, denoted $\mathcal{B}_k(I)$, is the set of all maximal configuration-spaces that a k-$\bar{\kappa}$-algorithm running on I could possibly observe.*

The observable configuration-spaces are determined as follows. For a given value of $\bar{\kappa}$ and starting configuration S, we can derive the sets of first-order adjacencies to S that are available to the algorithm (for the remainder of this section, it can be assumed that any mention of ith-order adjacencies is relative to S). Each choice of adjacencies is a product of which robots are coupled—for which every possible combination of moves will be represented in the adjacencies—and for the decoupled robots, which of their possible moves do they "autonomously" take. An algorithm only sees one of these sets of adjacencies. Which one it sees is *arbitrary*: for it to make a non-arbitrary, i.e. informed choice, it would need to be able to see and compare more than one set of adjacencies. This would require increasing $\bar{\kappa}$, which is invariant. In the same

fashion, given a choice of observable first-order adjacencies, we can derive the sets of second-order adjacencies that are available to the algorithm based on $\bar{\kappa}$. This process can repeat until adjacencies have been chosen for all possible configurations. $\mathcal{B}_k(I)$ is then the set of all possible configuration-spaces that could arise from this process.

Note the use of the phrase "*maximal* configuration-spaces" in Definition 4.1.1. What is meant by this is that the available choices of adjacencies assume that all coupled robots have all possible moves considered at every adjacency. Because we are seeking to prove claims about *all* algorithms with a given degree of coupling, we must assume that algorithms utilise the entirety of the information afforded to them by that degree. An algorithm may in practice observe a subset of one of these maximal observable configuration-spaces. This will be addressed further in Section 5.1.

4.2 Completeness

If it is possible that an algorithm will observe a configuration-space that does not contain a valid path, it cannot be complete. Based on this, an outline of a method of proving the incompleteness of MRMP algorithms with a given value of $\bar{\kappa} = k$ is as follows. Assume that a k-$\bar{\kappa}$-algorithm observes a configuration-space containing a valid path. Demonstrate a valid change, or finite sequence of changes, in the choice of adjacencies which would alter the observable configuration-space such that it no longer contains a valid path. If such alterations are possible, it is possible that a $\bar{\kappa}$-algorithm will observe the invalidated configuration-space, and so cannot be complete.

As a first demonstration of the proof idea, we will show that when $\bar{\kappa} = 0$, a solution cannot be guaranteed to be found even in a simple two-robot instance. Consider an instance with two robots starting diagonally to one another, and with targets requiring the robots to swap positions, as illustrated in Figure 4.1. When $\bar{\kappa} = 0$, every robot only considers one move. The observed configuration-space is essentially a path-graph. If it contains a valid path, the algorithm will return a valid solution. If not, it can't. Suppose that the algorithm picks out the valid path as illustrated in (a). An alternative choice of first-order adjacency can be found to make it such that the path is no longer valid, as illustrated in (b). Therefore, a $\bar{\kappa} = 0$ algorithm cannot guarantee to find collision-free plan for this instance. What if we increase $\bar{\kappa}$ to 1? Now there are two choices of sets of adjacencies (one for each choice of coupled robot), both of which allow for a valid path. The choice of adjacencies cannot be changed in such a way to avoid a valid path appearing in the observed configuration-space. $\bar{\kappa} = 1$ is enough to guarantee that a valid solution is returned for this instance, illustrated in (c).

Now, wielding this proof method in a more general fashion, we will provide a

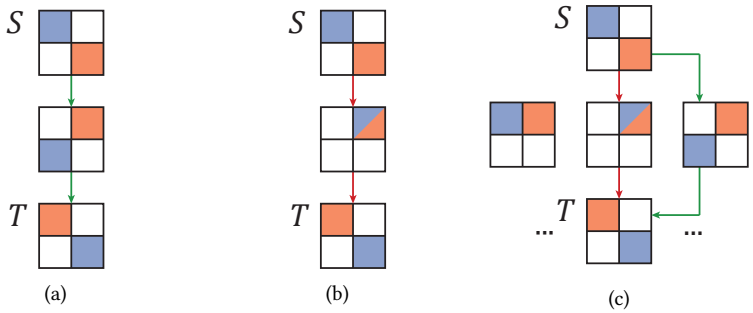


Figure 4.1: Configuration-space diagrams showing that $\bar{\kappa} = 0$ cannot guarantee to find a collison-free plan, even for a simple instance. (a) and (b) show valid and invalid choices of paths respectively which may occur with $\bar{\kappa} = 0$. (c) shows that increasing $\bar{\kappa}$ to one is enough to ensure that there is a collision-free plan in the observed configuration-space for this instance.

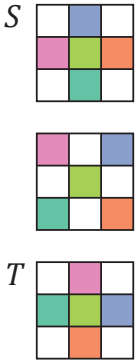


Figure 4.2: An optimal collision-free plan for the *cluster-of-five* scenario.

counter-example that shows that $\bar{\kappa} \geq 4$ is necessary for completeness. Consider the below scenario with five robots in a star formation. The targets of the four outer robots are such that each robot should move 90-degrees clockwise from its starting position, while the inner robot should finish in its starting position. We will call this the *cluster-of-five* scenario, and is illustrated in Figure 4.2.

First suppose $\bar{\kappa} = 0$, and an algorithm chooses adjacencies such that it finds the suggested route involving 90-degree clockwise rotations, as illustrated in (a) of Figure 4.3. An alternative choice of first-order adjacencies can be provided such that all robots move through the middle, causing a collision and invalidating the path, illustrated in (b). If $\bar{\kappa}$ is increased to 1, the adjacencies include alternative moves for *one* robot, allowing that robot to avoid the collision, but the remaining four robots still collide, illustrated in (c). $\bar{\kappa}$ needs to be increased to 4 in order to ensure that the collision in the centre can be avoided and a collision-free plan found, illustrated in (d).

In general, when five robots are competing for one position, at least four of them need to consider an alternative move. This is only possible if $\bar{\kappa} \geq 4$. This *cluster-of-five*

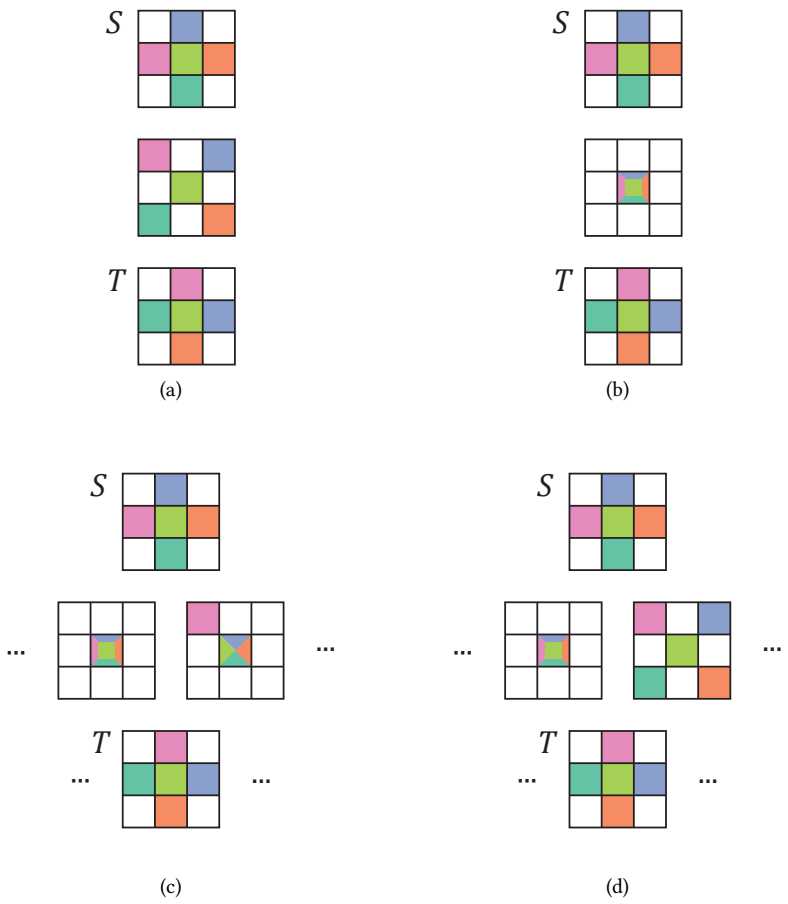


Figure 4.3: Configuration-space diagrams showing how the *cluster-of-five* scenario necessitates $\bar{\kappa} \geq 4$ in order to guarantee that there is a collision-free path through the observed configuration-space. (a) shows the collision-free path. (b) and (c) show when $\bar{\kappa}$ is too low, it is always possible to choose adjacencies such that there is no collision-free path. (d) shows that when $\bar{\kappa} \geq 4$, there is always a collision-free path in the observed configuration-space.

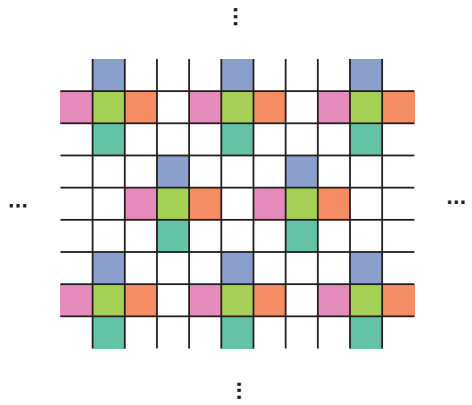


Figure 4.4: *many clusters-of-five* scenario. It is not shown, but is implicit in the definition of the scenario, that all robots want to move through the centre of their respective clusters, causing the pathological situation.

can be generalised to show that $\bar{\kappa}$ must be at least $\lfloor \frac{4}{5} n \rfloor$ for completeness in generality. Consider the scenario where n robots are divided into groups of five robots, where each group forms its own cluster-of-five scenario. If n is not divisible by 5, make a final smaller cluster—hence the floor in the bound on $\bar{\kappa}$. Call this the *many-clusters-of-five* scenario, illustrated in Figure 4.4. In each cluster, all but one of the robots must be coupled in order to consider alternative moves and ensure the presence of a valid path in the observed configuration-space. Therefore $\bar{\kappa} \geq \lfloor \frac{4}{5} n \rfloor$ is necessary for completeness. A final extension of the counter-example allows us to strengthen the lower bound on $\bar{\kappa}$ to $n - 1$ and arrive at our final claim relating coupling and completeness.

Theorem 4.2.1. *$\kappa \geq n - 1$ is necessary for an algorithm to guarantee completeness across all instances.*

Proof. Suppose that in the many-clusters-of-five scenario, the robots that remained decoupled have targets such that they move on to form their own clusters of five, requiring that four fifths of those robots also need to become coupled. This process can be iterated until a scenario is reached where $n - 1$ robots need to be coupled for there to be a valid path in the observed configuration-space: every robot except for one robot in the final cluster, which may remain decoupled. □

4.3 Optimality

The cluster-of-five scenario can also be used to show that $\bar{\kappa} = n$ is necessary for optimality.

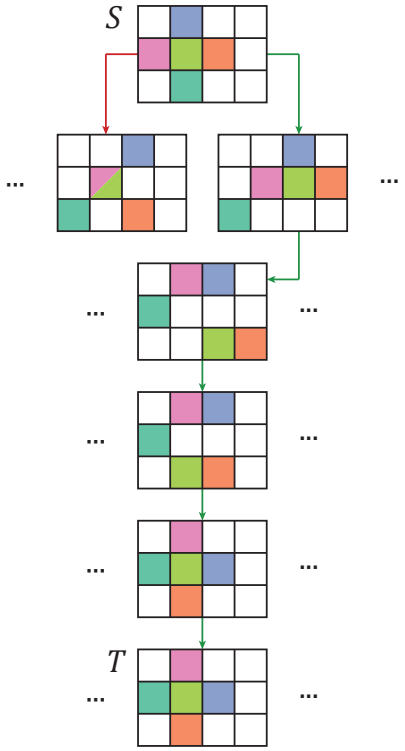


Figure 4.5: Illustration showing how $\bar{\kappa} = 5$ is necessary to guarantee that an optimal makespan path is in the observed configuration-space in the *cluster-of-five* scenario.

Theorem 4.3.1. *$\bar{\kappa} = n$ is necessary for an algorithm to guarantee optimality across all instances.*

Proof. Suppose that in the cluster-of-five scenario, $\bar{\kappa} = 4$, and instead of all of the four outer robots becoming coupled, the inner robot becomes coupled while one of the outer robots remains decoupled. A valid path can still be found, but now the robot opposite the decoupled robot must move outwards (assume we have space in the grid), making space for the inner robot, which is itself making space for the decoupled robot, insistent on taking the route through the middle. After four moves, the two robots which made space for the decoupled robot can finally move onto their targets. With $\bar{\kappa} = 4$ in this scenario, it is possible that a valid but suboptimal solution is returned. $\bar{\kappa} = 5$ is necessary for optimality. This is illustrated in Figure 4.5. Using the same method as above of grouping n robots into many clusters of five, it follows that $\bar{\kappa} = n$ is necessary for optimality in general. □

4.4 On "Arbitrary Choice"

Above I alluded to the idea that the choice that an algorithm makes of the possible adjacencies available to it and by extension which of its observable configuration-spaces it observes is *arbitrary*. The given proofs of the lower bounds on $\bar{\kappa}$ necessary for completeness and optimality depend on this idea, so it is worth taking some time to clarify and justify it.

The cluster-of-five scenario used above is by all means contrived. This might lead one to question the generality of the proofs—why shouldn't we expect an algorithm to avoid this scenario, especially given that we know it to be a compromising position? Let's say that a $\bar{\kappa} = 0$ algorithm presented with the cluster-of-five scenario finds a valid path. This could certainly happen, but how could it be guaranteed to happen for *all* $\bar{\kappa} = 0$ algorithms? There would need to be a way to ensure that all outer robots avoid the middle position. But all robots are decoupled—they are autonomously making their way to their targets. Already, the idea that we could somehow *guarantee* avoiding the collision without coupling is at-least conceptually inconsistent. Yes, for any given algorithm it does ultimately make the decision on how every robot moves, even those that are decoupled. But the decisions an algorithm makes for a decoupled robot do not, by definition, take into account the greater system. The observable configuration-spaces encapsulate that fact. Any given member of the observable configuration-spaces plays out a hypothetical scenario: say that these $\bar{\kappa}$ robots are coupled, and the rest are not. Now say that an algorithm decided on *this* path for *this* decoupled robot, and *that* path for *that* one. If that is how an algorithm proceeded, *here* is everything it could know (observe). And from that we can see every decision the algorithm could make. This could be framed as the subset of the solution-space that is *accessible* to an algorithm observing *this* configuration-space. The *observed* configuration-space—which of the observable ones an algorithm actually plays out—is *retrospective*. It is the aggregate of everything an algorithm has seen at the *end* of its execution. If it has not seen a valid path, it cannot return one. Likewise, if it has not seen an optimal path, it cannot return one. Certainly we can have an algorithm which "notices" that the choice of a decoupled robot is problematic and so corrects it. But doing so requires firstly being able to see an alternative, and secondly having control over the decision-making of the robot *after the fact* so that it can take that alternative. Both of these actions not only require, but are definitive of, a higher degree of coupling.

So, when speaking generally about all k-$\bar{\kappa}$-algorithms for a specific value of k, we know that for any given instance there is a set of configuration-spaces that these algorithms can observe. We know that which of those configuration-spaces a spe-

cific algorithm will observe is in a sense *arbitrary*—to pick and choose between the available choices in an informed way would necessitate a higher degree of coupling. So we cannot make *a priori* claims about which of the possible configuration-spaces will be observed. So if it is possible to observe a configuration-space which does not contain a valid path, or which does not contain an optimal path, it cannot be ruled out that such a configuration-space *will* be observed. And if it *is* observed, the lack of a satisfactory path can only be remedied by observing more configurations, which, again, would require the algorithm to have a higher degree of coupling. This is why we can place limits on completeness and optimality based on $\bar{\kappa}$.

Chapter 5

Empirical Analysis of Coupling

The previous chapters have directly related the degree of coupling to the dimensionality of the observed configuration-space. Therefore, the size of the observed configuration-space grows exponentially with respect to κ. From there, it is no great leap to suggest that increases in coupling during an algorithm's execution will have a detrimental impact on its run-time. This chapter is focused on empirically measuring how much coupling actually occurs in M* and CBS, and its impact on performance. Before going into the experimental results, we look into how coupling might be measured at a more precise level than the course-grained view presented above.

5.1 Finer-grained Measurements

The observable configuration-spaces induced by a particular value of $\bar{\kappa}$ are *maximal*: they assume that for every coupled robot, all of its possible moves are considered from every configuration. This is useful for proving theoretical limits based on $\bar{\kappa}$ because we can make claims along the lines of "even if an algorithm uses all of the information and choice afforded to it by its degree of coupling, it cannot necessarily do this or that". In practice, however, an algorithm might not use all of that information and choice—it might observe a *subset* of one of the observable configuration-spaces. As such, it will be useful to complement κ and $\bar{\kappa}$ with finer-grained measurements for quantifying to what extent coupling occurs during a run of an algorithm on a particular instance.

Average Degree of Coupling

The average degree of coupling can be defined, with respect to a given observed configuration-space, as the average number of robots that have multiple moves considered across the adjacencies of all configurations in the observed configuration-space. This measurement can be compared against the overall degree of coupling for that observed configuration-space, as well as the maximum degree of coupling for the algorithm used, to gauge how much coupling actually occurred throughout the execution of the algorithm, compared to what we would expect if the algorithm had used all of the information and choice afforded to it by its degree of coupling. An algorithm with a lower average degree of coupling while still providing the guarantees that it is able to based its overall degree of coupling is desirable: it indicates that it traverses a smaller subset of the total observed configuration-space, and thus would be expected to run faster, all other things being equal.

Local Coupling

The coupling defined by κ can be considered global or "robot-to-system" coupling, in that once a robot is coupled, its decision-making is abdicated to the system as a whole. We may also wish to define robot-to-robot or *local* coupling: coupling which occurs between a pair of robots. On this view, locally coupled robots share information and collaborate in decision-making, but from the perspective of other robots and the system as a whole, are decoupled. Where global coupling might be represented as a boolean function which is true for coupled robots and false for decoupled, local coupling must be represented as a "coupling graph" where there is an edge between pairs of coupled robots. The degree of local coupling is parameterised by n, and has a maximum value of $\frac{n(n-1)}{2}$, corresponding to a complete coupling graph and equivalent to a global degree of coupling of n.

Clearly, local coupling provides more precise information than global coupling, and may well be a useful consideration in the design of MRMP algorithms. It has not been emphasised above because it is not useful in evaluating the limits of MRMP algorithms in terms of their degree of coupling. Firstly, it is difficult to see how an algorithm-independent means of quantifying local coupling could be achieved. We certainly cannot determine it from observed configuration-spaces alone: say that a set of adjacencies in an observed configuration-space reveal that a pair of robots are coupled. How are we to know whether that coupling has occurred at a local or global level from this information alone? Further information is needed, and it is likely that the information required will need to be specific to how local coupling manifests itself in a specific algorithm. Secondly, if we used local coupling as the parameter against

which guarantees such as completeness and optimality can or cannot be made, we would ultimately converge on the necessity of global coupling (in the worst case): in the cluster-of-five scenario, given any permutation of local couplings where the outer robots are not completely coupled with one another, we can conceive a scenario where a collision occurs.

With that said, local coupling is certainly a useful notion. An algorithm which dynamically couples at the local level may be able to find optimal solutions in practice while searching a smaller subset of the total configurations-space than an algorithm that dynamically couples at the global level. It is reasonable to expect that effective use of local coupling would result in a lower average degree of (global) coupling.

Time-local Coupling

Observing solutions to MRMP instances play out, it becomes clear that often a particular pair of robots only need to interact during a part of their respective journeys. This motivates making the notion of local coupling even more precise by adding the dimension of time. The coupling graph becomes a temporal graph, where a pair of robots are coupled at time t if there is an edge between them at time t. Unlike global and local coupling, we cannot parameterise the total amount of time-local coupling that occurs purely in terms of n, as the number of time-steps will depend on the makespan of a specific instance. We may however analyse it in terms of the minimum, average, and maximum degrees of local coupling that occur at any given time. Like local coupling, taking time-local coupling into consideration in the design of MRMP algorithms may help to further "tighten" how much coupling actually occurs.

Of the above, average degree of coupling is what will be measured in the experiments that follow, since, as has been discussed, local and time-local coupling are not necessarily measurable retrospectively. It was nonetheless useful to consider them to deepen our understanding of how coupling can play out. Furthermore, they are factors which may ultimately influence the average degree of coupling of a particular algorithm running on a particular instance.

5.2 Experimental Results

Experiments were run in an effort to answer the question set RQ1 listed in Chapter 2, repeated here.

RQ1:
1. How much coupling actually occurs when running dynamic fully coupled algorithms?

2. What is the relationship between actual coupling and run-time?
3. How does the run-time of dynamic fully coupled algorithms compare to that of static fully coupled algorithms?
4. How does the occurrence of collisions impact actual coupling and run-time?

5.2.1 Setup

CBS, M*, and A* were tested on 8×8-grid instances with n ranging from 4 to 16. A timeout of 10 minutes was set. For each value of n, there were 5 instances generated for each cluster-factor of $0, 0.33, 0.66$, and 1, resulting in 20 instances per n, and 260 total instances. For each instance solved by CBS and M*, the degree of coupling in the observed configuration-space: κ, and the average degree of coupling in the observed configuration-space: $\hat{\kappa}$, was measured. These values could be directly measured for M* as it directly searches the configuration-space. For CBS, the implicitly searched configuration-space was constructed from the constraint-tree in order to measure coupling. Furthermore, for each instance, the number of collisions occurring in a "naive solution", that is, routing each robot according to its shortest path, completely ignoring other robots, was measured.

5.2.2 Hypotheses

It is expected that dynamically coupled algorithms will, for most instances, have an *observed* degree of coupling (denoted κ) lower than n (RQ1.1). It is also expected that, because CBS goes about coupling in a more "fine-grained" way, that it will have a lower *average* degree of coupling (denoted $\hat{\kappa}$). Because, as we have seen, the size of the observed configuration-space grows exponentially with actual coupling, it is in turn expected that lower coupling will be associated with lower run-times (RQ1.2). For the same reasons, it is expected that the dynamically coupled algorithms will have improved performance compared to the static-fully-coupled A* (RQ1.3). Increases in coupling in both CBS and M* are directly associated with collisions. It is therefore expected that instances where the naive solutions experience more collisions will force more coupling and higher run-times (RQ1.4).

5.2.3 Results

The experiments showed that dynamic coupling significantly improves performance over the static-fully coupled A*. The ability of A* to solve instances in under 10 minutes immediately dropped off for $n > 5$. M*, however, was only marginally better than

A*, dropping quickly off after $n > 8$. CBS however performed very well, only slowly starting to drop off for $n > 8$ before completely dropping off for $n = 16$. Figure 5.1 visualises these results: (a) shows the run-times, and (b) the success-rates (i.e. the number of instances solved in under 10 minutes).

The significant improvement in the performance of CBS correlates with the coupling measurements. The observed degree of coupling in CBS was notably lower than M* (relative to n). Furthermore, the average degree of coupling in CBS was notably lower than the observed, indicating that the maximum observed degree of coupling only actually occurs across a few adjacencies. The average degree of coupling for M*, while still lower than the observed, was closer compared to CBS. Figure 5.2 visualises these results.

There was a very strong correlation between collisions and run-time. Independently of n, more coupling on average occurred in CBS and M* on highly clustered instances, which corresponded with higher run-times. Figure 5.3 visualises these results.

5.3 Conclusions

Throughout the previous chapters, we have defined coupling in terms of the dimensionality of the observed configuration-space, the size of which grows exponentially with dimensionality. Therefore, it was expected that if dynamically coupled algorithms can find a solution while keeping coupling low, they will have favourable run-times. This has been confirmed by the above experiments. Particularly satisfying is just how strongly observed coupling correlates with run-time not only within each algorithm, but across algorithms: in Figure 5.2.a, it can be seen that $\kappa \approx 6$ for both M* when it dropped off at $n = 8$, and for CBS when it dropped off at $n = 16$. Given that CBS does not explicitly search the configuration-space, rather the observed configuration-space was recreated post-hoc in order to measure κ, this provides some empirical validation of the notion of an *implicit* search of the configuration-space that was used to justify the definition of coupling given in Chapter 3 as a compelling algorithm-independent measure.

The correlation between collisions, coupling, and run-time, especially considering that the correlation is completely independent of n, drives home what might ultimately be the salient point of this part of this thesis: the need to avoid collisions is what makes MRMP hard, and if we want to *always* avoid *all* collisions, we need to couple (Theorem 4.2.1). But the more we couple, the harder the search becomes. This is what was formalised in Chapters 3-4, and empirically verified here.

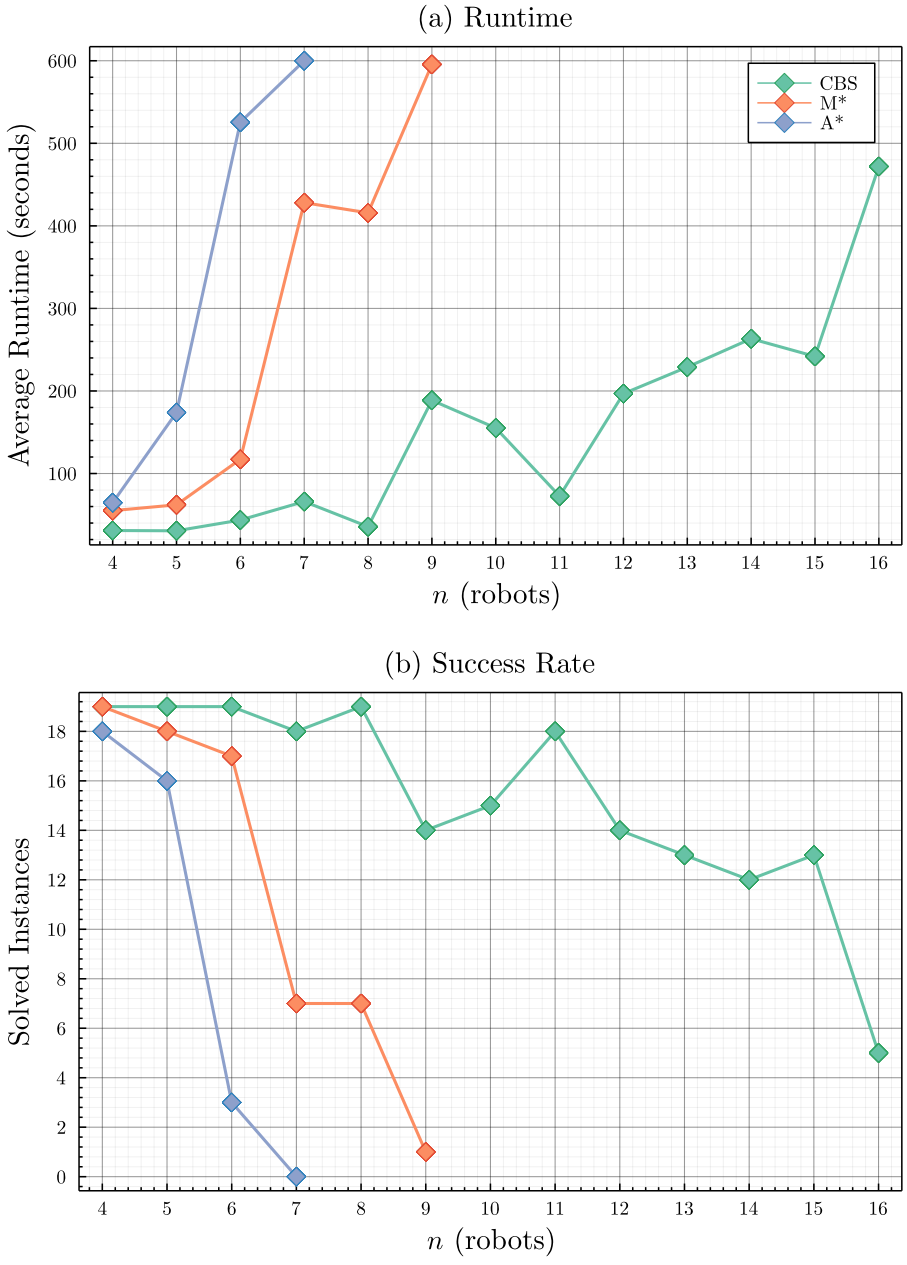


Figure 5.1: Average run-times (a) and success-rates for CBS, M*, and A*.

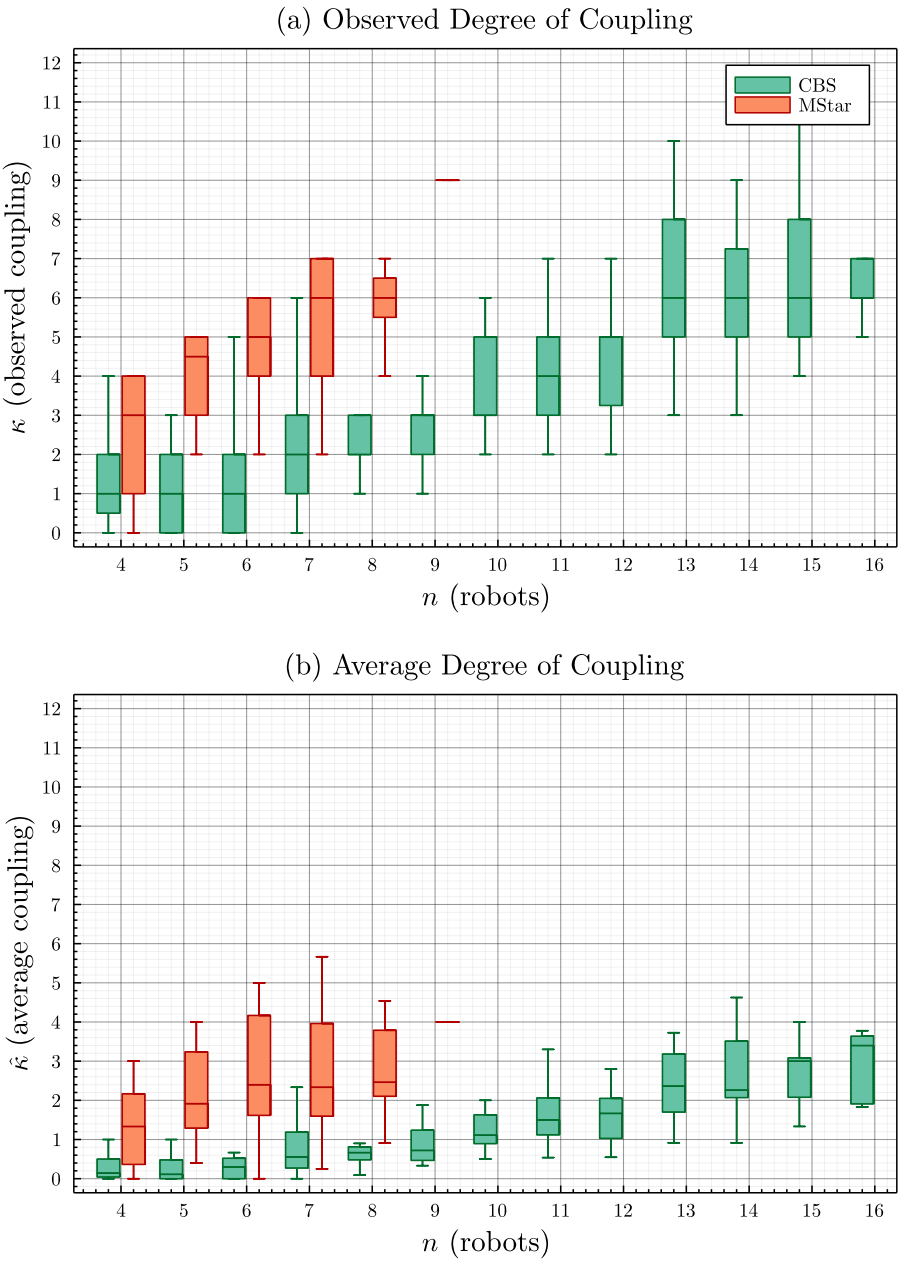


Figure 5.2: Observed (a) and average (b) degrees of coupling measured for CBS and M*.

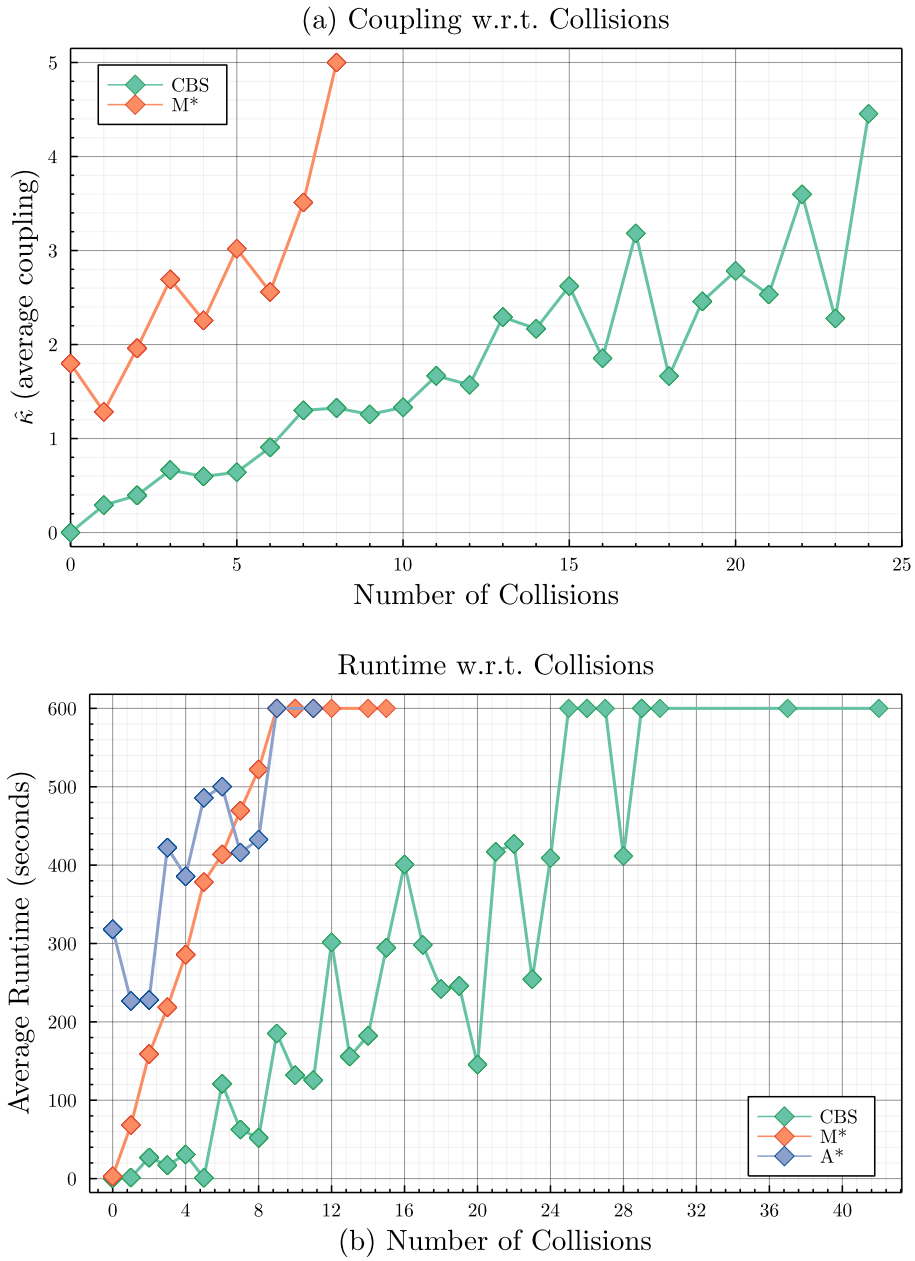


Figure 5.3: Correlation between collisions, coupling, and run-time. Note that in (a), only numbers of collisions up to 25 are shown as for larger values, no solutions were found, and coupling was not measured in these cases. On the other hand, run-time was still measured, so the higher values are displayed in (b).

Part III

MRMP as a Complex System

Chapter 6

Complexification

Complex systems are characterised by the behaviour of the system as a whole being dependent on the interactions of its components. We cannot apply a reductionist approach—analysing the system as a whole by analysing its individual components in isolation—to such systems. Yet, most models of complex systems tend to assume that they are *structurally static*: the number and nature of interactions between components remain unchanged throughout a system's life-cycle. Through this assumption, reductionism has, in the words of Malkin, "snuck in through the back door" into the study of complex systems: they are analysed by analysing the many individual static structures that they might take on [19]. Observations show that real-world complex structures do not take on such static structures. Rather, they are *structurally dynamic*. This chapter explores the relationship between *gradual complexification* in complex systems—one form of structural dynamicism—and dynamic coupling in MRMP. The first section introduces the relevant concepts from the study of complex systems. The second section explores the concept of complexification, before we consider its analogy with dynamic coupling in the third. This analogy is used to motivate a fitness function for MRMP such that it can be framed as a complex system exhibiting gradual complexification.

6.1 Complex Systems

The study of complex systems encompasses a wide range of fields and phenomena. As such, there is no single precise definition of what constitutes a complex system, or of how complexity in this context is quantified. Especially since the concept is being introduced within a work in which computational complexity is also considered, it will be important to pin down what is meant by the complexity of a *system* in this

discussion.

A system has a corresponding *fitness* which we want to maximise. A system is complex if it is made up of two or more components which can interact with one another in such a way that the fitness of the overall system is dependent on the interactions between the individual components. In contrast, a "non-complex" system would be one where the fitness is *linear additive*: the fitness of the system as a whole can be directly determined by the sum of the individual fitnesses of its parts. With this in mind, the term "complex system" can be considered a concise and arguably more compelling synonym for "non-linear dynamical system".

Complexity

For our purposes, we will quantify the complexity of a system by the number of interactions among its components. So, in the context of MRMP and coupling, each robot is a component, and they interact if they have become coupled. Then, the complexity of a system of robots is quantified by how much coupling there is. In other words, the degree of coupling directly corresponds to complexity. In other contexts, more than simply the number of interactions may contribute to complexity. The *intensity* and *type* of interactions may also play a role, for example. For our purposes, this straightforward quantity-based definition is sufficient.

Evolvability and The Fitness Landscape

The fitness of a complex system at any time is determined by its state at that time, which in turn is determined by the states of its components, and the interactions between them. The *fitness landscape* is then a representation of all possible states mapped to their fitnesses. A problem modelled as a complex system becomes an optimisation problem where the goal is to find the global maximum of a fitness function f. We want to *evolve* the system such that it finds the *peaks* and avoids the *valleys* in the fitness landscape. The *evolvability* of a system is its ability to adapt into a fitter state—or find the peaks in the fitness landscape—over time. Thus systems that are more evolvable with respect to a given fitness landscape are desirable.

6.2 Gradual Complexification

Gradual complexification is the gradual increase in the complexity of a system over time, and is one way in which a system might be structurally dynamic. Observations show that most complex systems in the real-world exhibit such behaviour. Some of the more obvious examples follow. Biological life began with simple, self-replicating

molecules that gradually increased in complexity, leading to the vast diversity that we see today. Economics began with straightforward means of managing the exchange of goods and services at the local level, before gradually growing and coalescing into the massively interconnected global financial system. Languages began with small sets of sounds and symbols, gradually growing into massive vocabularies accompanied by complex grammars.

Malkin identified that while practitioners of complex systems are aware that complexification occurs, rarely is it taken into account in models that evaluate them in terms of fitness landscapes [19]. Experiments comparing the eventual fitness of systems exhibiting gradual complexification versus equivalent systems beginning in a complex state revealed that gradual complexification *increased evolvability*: such systems were more likely to evolve into the global maximum of the fitness landscape. This indicates that not only is complexification interesting because it appears to be what actually happens in the real world, but also because it appears to be useful in—to use the language most relevant to our purposes—finding the best solution in a complex search-space.

To gain an intuition for the power of gradual complexification, it will be instructive to consider the two extrema of the alternative: a statically simple system and a statically complex system. The systems can be perceived as search-spaces, where an increase in complexity corresponds to an increase in the dimensionality of the space. The simple, low-dimensional search-space will be easy to search. But it is unlikely that it will contain the best solution. The complex, high-dimensional search-space is likely to contain the best solution, but it will be very difficult to actually find that solution if we are dealing with more than a handful of dimensions. The low-dimensional option exemplifies the pitfalls of a greedy search strategy: we prioritise a few routes based on minimal information, but we have no certainty that what looks like the best route now will indeed be the best in the long run. The high-dimensional option exemplifies the pitfalls of a brute-force search strategy: there is theoretical certainty of finding the best solution, but the probability of finding it in a reasonable time-span rapidly diminishes as we consider more complex search-spaces. Returning to the notion of fitness, the simple system is likely to be limited to local maxima: it may not have a route to the highest peak in its fitness landscape. The complex system has *low evolvability*: the highest peak may well be there, but it will be very difficult to actually reach it if we aren't lucky enough to start nearby.

Gradual complexification allows us to get the best of both worlds. We start with a simple system based on a few assumptions, i.e. low dimensionality. Then, throughout our search, we systematically *complexify*. This can be visualised by organising the fit-

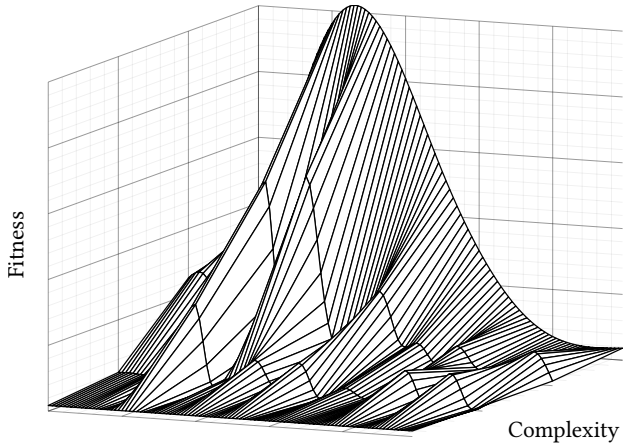


Figure 6.1: A dynamic fitness landscape.

ness landscape with the dimension of complexity. At low complexities, the landscape is likely to be mostly smooth with some low peaks. As complexification occurs, the fitness landscape becomes more complex, and one or more distinct, higher peaks are likely to emerge. Such a landscape will be termed here as a *dynamic fitness landscape*. Figure 6.1 visualises a dynamic fitness landscape. Lotto calls it an "uber-landscape", and from this perspective describes the finding of Malkin's experiments (that gradually complexifying systems are more evolvable) as "walking along the ridges connecting the peaks in the fitness landscapes" that make up the dynamic landscape, in doing so avoiding the valleys [18]. Through gradual complexification, we can leverage the evolvability of a simple system while still eventually reaching the peaks of a complex system.

6.2.1 Efficiency, Creativity, and Innovation

If we take creativity to be connecting ideas that have not been connected before, then creativity in an evolving system would be a transition between states that has not been made previously. A complex system has greater capacity for creativity because it can "see" more possibilities. Efficiency on the other hand can be thought of as taking the clearest route to a payoff (i.e. an increase in fitness). Efficiency and creativity appear to be decidedly at odds with one another: a route through unexplored territory cannot be considered clear. But it is embracing both and managing the trade-off between the two from which *innovation* emerges. An innovative individual or company is highly open to exploring many possibilities, but also knows when to focus in on one. On the other hand, pure creativity may be paralysed by possibility, while pure efficiency risks missing out on greater possibilities in favour of certain, short-term gain.

Innovative Silicon Valley companies may be considered the archetypal embodiment of this trade-off between creativity and efficiently of our time, but in fact they are mimicking behaviour exhibited by natural complex systems as old as life itself, in particular by biological evolution and the brain. Malkin's experiments further showed that beyond mere gradual complexification, systems which embraced "creativity" by complexifying with a degree of randomness, while maintaining efficiency by seeking to minimise their "energy state" were the most evolvable [18, 19]. For our purposes, minimising the energy state can be thought of as, given two states of equal fitness, prioritising the one that can be arrived at with the least complexification. A system exhibiting this behaviour will occasionally *decomplexify*: if it sees a less complex route to an equivalent quality solution, it will take it.

6.3 Complexification and Dynamic Coupling

Anyone who has read this work in its entirety up to this point can be forgiven for finding the description of gradual complexification tiresome: it almost directly maps to how we defined coupling, which itself almost directly mapped to the idea of sub-dimensional expansion in M*. Clearly, coupling is a form of complexification, and the behaviour of dynamically coupled algorithms mirrors gradual complexification. This opens up the intriguing possibility of using this analogy to motivate approaches to MRMP.

6.3.1 Collisions and Chaos

The two common MRMP objectives: makespan and total-distance, can be thought of as seeking to optimise for the physical quantities of time and energy, respectively. In seeking to avoid collisions, the process of solving an MRMP instance is also implicitly minimising a third physical quantity: *entropy*. A plan with many collisions is chaotic, i.e. high entropy, while a collision-free plan is ordered, i.e. low entropy. Making this explicit will allow us to treat MRMP as a complex system. On this view, a solution with fewer collisions has a higher fitness. This can be formalised with *relative entropy*.

Relative entropy, or Kullback-Leibler divergence, measures the *divergence* of one probability distribution from another.

$$D_{KL}(P \mid\mid Q) = \sum_k P_k \log_2 \frac{Q_k}{P_k} \tag{6.1}$$

D_{KL} has the property that it is always greater than or equal to zero, and equals zero only when $Q = P$.

For our purposes, we want P to be a probability distribution which characterises the probability of a robot encountering a collision in a specific plan for an MRMP instance, and Q to characterise the probability of a robot encountering a collision in some baseline plan where collision probability is maximised—it makes sense that we want to maximise divergence from such a chaotic plan. Because these are just single-variable, boolean-valued distributions representing "collision" or "no-collision", we will define them in terms of collision probabilities p and q.

For some plan with c collisions and a makespan of t, the collision probability can be defined by supposing that the occurrence of collisions is uniformly distributed throughout the plan. Then the probability of any one robot encountering a collision in any one move is:

$$p = \frac{c}{nt} \tag{6.2}$$

This is sensible as the expected number of collisions throughout the execution of a plan given this distribution is recovered by $nt \cdot p = c$.

q can be defined independently of any specific plan or instance by considering an extreme scenario where *every* position on the grid contains a robot. Then for every robot, only one of its five possible moves—remain—avoids a collision. So without any additional information on robot behaviour, the collision probability of any one move is $q = \frac{4}{5}$. Now we can define a fitness function in terms of the divergence of p from q:

$$D(p) = p \log_2 \frac{p}{q} + (1 - p) \log_2 \frac{1 - p}{1 - q} \tag{6.3}$$

$$= p \log_2 \frac{5p}{4} + (1 - p) \log_2 5(1 - p) \tag{6.4}$$

The case where there are no collisions, and so $p = 0$, may appear problematic because $\log_2$ is undefined at zero, but $p = 0$ should in fact maximise the fitness function. However, notice that the left term will simply be zero in this case due to the outer multiplication by p. Then the maximum divergence, occurring when there are no collisions, is $\log_2 5$. This function has the other key properties we would want and expect: it equals zero only in the worst-case scenario where $p = q$, and increases monotonically thereon.

Why not just determine fitness in terms of makespan or total-distance, since these are the objectives that MRMP seeks to minimise? Because we want to characterise solutions in terms of the degree of coupling necessary to arrive at them. Solutions arrived at without coupling, but by simply taking a shortest path to target for each robot, would always have the highest fitness if it was determined by the objective. But for

non-trivial instances, such solutions will typically have many collisions. Minimising collisions, or entropy, means that feasible solutions will form the highest peaks in the fitness landscape. Then, coupling (κ) is the dimension along which we complexify, forming the dynamic fitness landscape. There will always be a peak where $\kappa = n$, but what is interesting is the (potential) presence of peaks for lower values of κ. An algorithm designed to find these peaks could in principle be expected to find the best solution possible while keeping coupling as low as possible. Keeping coupling low would in turn be expected to keep run-time down and evolvability high. Augmenting the fitness function such that solutions with a lower degree of coupling have a higher fitness would contribute towards this goal. Then, an algorithm looking for the peaks in the resulting dynamic fitness landscape is encouraged to *decomplexify* when appropriate by prioritising low-coupling solutions. Finally, we can reincorporate makespan or total-distance into the fitness function such that the absolute highest peaks in the landscape correspond to the optimal, collision-free solutions, arrived at with as little coupling as possible. These considerations motivate us to augment (6.4) slightly to arrive at our final fitness function:

$$f(P) = D(p_P) \cdot \sigma\Big(\frac{n - \kappa_P}{n}\Big) \cdot \tau(b_P) \tag{6.5}$$

Where P is a plan, p_P is the collision probability of P as defined above, and κ_P is the degree of coupling used to find P. σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, which as used penalises plans arrived at with higher degrees of coupling. b_P is the objective value of P, be it makespan, total-distance or, some other. τ is a placeholder to be instantiated with a function which should penalise high objective values, similarly to how σ is penalises coupling. Figure 6.2 visualises f disregarding the τ term. The shape of the function is desirable: the number of collisions dominates the output, while lower degrees of coupling are rewarded, resulting in a global maximum where there are no collisions and no coupling.

Figure 6.3 visualises the dynamic fitness landscape induced by this fitness function for a randomly generated five robot instance. (a) shows the landscape if just relative entropy is used to determined fitness; (b) shows how the landscape is changed by penalising high coupling. In both, it can be seen how "walking along the ridges connecting the peaks" could play out as gradual complexification occurs: there are smooth ascending paths towards peaks in the landscape when we start at $\kappa = 1$ then complexify. Compare this to if we started at (say) $\kappa = 3$. There are peaks to be found there, but the landscape is rugged and if we don't start nearby, it is likely we will get stuck in a local maximum. Or if we started at $\kappa = 5$: as expected, a collision-free solution can be found there, but the landscape shows that it can be found with less

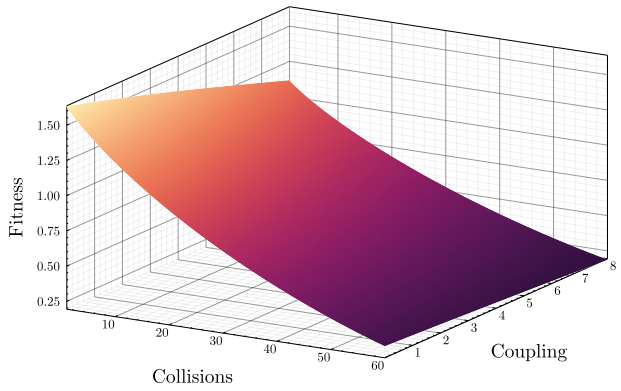


Figure 6.2: Visualisation of the fitness function f defined in 6.5. The data used to create the figure assumes $n = 8$ and a makespan of 16 in determining the collision probability p.

computational effort by exploring the low-coupling regions. (b) also clearly shows the result of penalising coupling: now there are distinct peaks where the best (or something close to the best) solution can be found with $\kappa = 2$ or 3, and incentive is given to look for it there instead of in the high-coupling regions.

It is worth acknowledging that a fitness function with the desired shape and behaviour probably could have been arrived at without the above fuss about relative entropy. But it is satisfying, and validating of the concepts which lead us to treat MRMP in this way to begin with, to see that the desired function emerges naturally by applying those concepts.

An popular account of the ideas on complexification that motivated this chapter can be found in Chapter 10 of *Deviate: The Science of Seeing Differently* by Beau Lotto, presented in the context of a neuroscientific account of human perception and creativity. The theoretical background along with the referenced experiments can be found in David Malkin's PhD thesis: "The Evolutionary Impact of Gradual Complexification on Complex Systems".

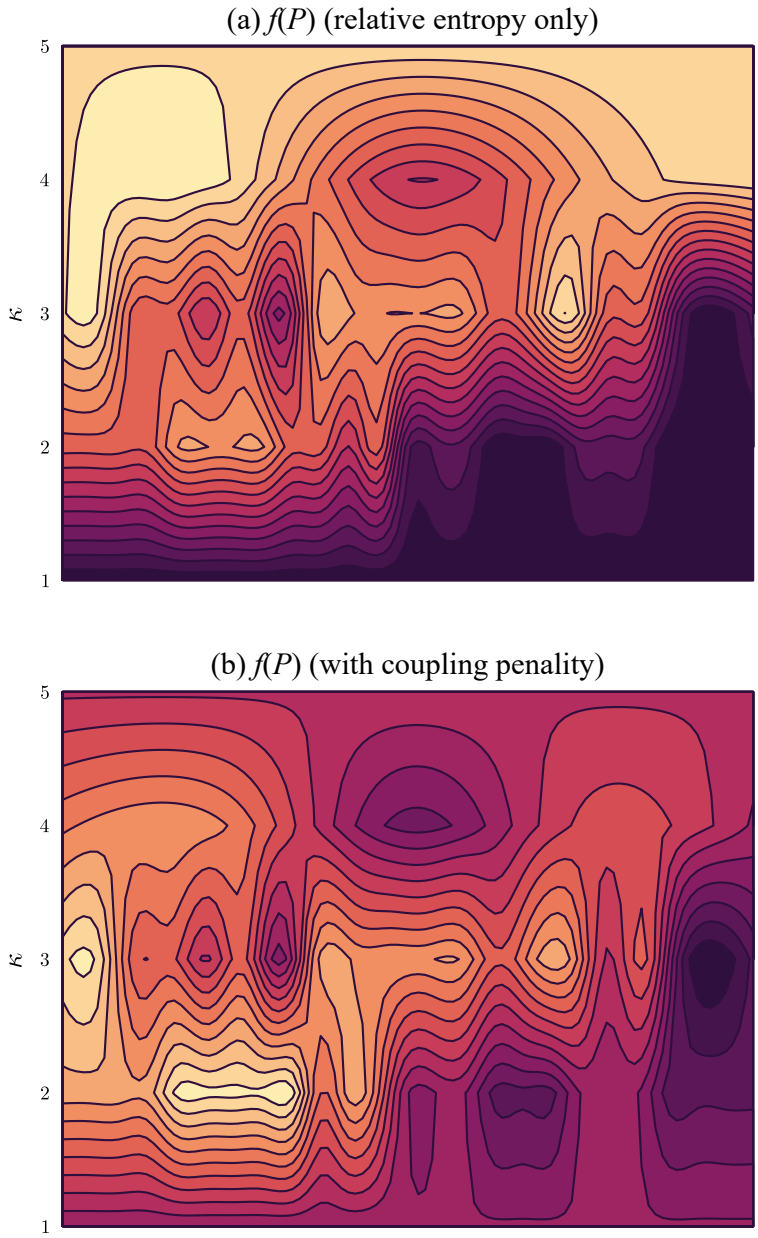


Figure 6.3: Contour plots visualising the dynamic fitness landscape induced by the designed fitness function. Brighter colours indicate higher fitness. (a) excludes the coupling penalty, thus is just the relative entropy, while (b) shows the impact of penalising coupling. The data is based on a randomly generated five robot instance. The fitnesses were arrived at by creating all possible partitionings of five robots. κ for each partitioning is determined by the size of its largest partition. Then a solution for each partitioning was found by solving each partition independently using CBS, and merging the results into a final plan to be passed into the fitness function. The partitions within each value of κ are ordered lexicographically. Putting all of this together results in the landscapes shown.

Chapter 7

Genetic Algorithms

This chapter provides a brief introduction to genetic algorithms, in order to serve as background to the next chapter, which introduces a genetic algorithm for MRMP, motivated by the previous. The exposition of genetic algorithms that follows should be considered "just enough" for the purposes of this work. Mitchell [20], among many others, offers a thorough introduction to the topic. It should also be noted that genetic algorithms fall under the broader field of *evolutionary algorithms*, and should not be construed with other techniques from the field such as evolutionary programs and genetic programs which,h while also motivated by biological evolution, operate differently.

7.1 Overview

A genetic algorithm (GA) is a type of metaheuristic that uses operations motivated by biological evolution, such as natural selection, crossover, and mutation, in order to quickly find the best or "fittest" part of a search-space. To solve a problem using a GA, first we must be able to encode elements of its solution-space as a *chromosome*. Typically, a chromosome will be a bit-vector where each entry indicates whether or not a particular parameter is active, although certain problems and GA's may call for more complex encodings. We also need a fitness function f over the solution-space, which we will seek to maximise. With an encoding and fitness function in hand, a typical GA will proceed, broadly, as follows:

1. *Initialisation*: Begin with an initial population of size s, randomly sampled from the space of possible chromosomes.

2. Evaluate the fitness, $f(x)$, of each individual x in the population.

3. Repeat until we arrive at a new generation of size s:

   (a) *Selection*: select a pair of individuals from the population, weighted such that those with higher fitnesses have a higher probability of being selected.

   (b) *Crossover*: with probability p_{cross}, create two new children by performing crossover on the chromosomes of the selected pair. If crossover does not occur, the chromosomes of the selected pair are taken unchanged into the next step.

   (c) *Mutation*: With probability p_{mut}, perform mutation on the chromosomes resulting from the previous step.

   (d) Add the resulting chromosomes to the new generation.

4. Repeat steps 2 and 3 until either a specified minimum fitness threshold has been reached, or a specified maximum number of generations have been iterated. At this point, output the individual in the population with the highest fitness.

Essentially, our fitness-landscape is a mapping of chromosomes to fitnesses, and our goal is to find the chromosome corresponding to the highest peak in the fitnesses landscape. A GA operates under the assumption that combining the properties of two good solutions will, on average, produce a better solution, thus providing a route to the peak. Sections 7.4 and 7.5 provide further discussion on when and why this approach works. With this framework in mind, further details on each of the key steps of a GA: initialisation, selection, crossover, and mutation, are provided below.

7.2 Genetic Operations

Initialisation

Typically, the initial population will be taken from a uniform sample of the solution-space, although it may be desirable to weight the sample towards chromosomes that are more likely to have a higher fitness. This carries the risk of premature convergence, causing the population to get stuck in a local maxima in the fitness landscape.

Selection

There are a number of possible selection methods with various trade-offs. The most straightforward approach is to make selection probability directly proportional to fitness. This results in what is known as "roulette wheel selection". A few problems may arise with this approach. If the absolute differences between the best and worst

fitnesses are small, then inferior chromosomes will have a very similar chance of reproducing as the fittest. If the absolute difference is too large, then it is probable that the best few chromosomes will parent most of the next generation, leading to a lack of diversity in the population and premature convergence to a local maximum in the fitness landscape.

The latter issue may be resolved by *stochastic universal sampling*: whereas in the first approach the roulette wheel is spun each time we choose a pair of parents, in this approach we spin the wheel once, but have many pins evenly spaced around the wheel: one for each parent to be chosen. Using this method, the fittest individuals are still more likely to be chosen more often, but less fit individuals still have a chance to be selected as the spacing of the pins ensures that one very fit individual cannot be chosen repeatedly across many samples.

The first issue—of small differences in absolute fitness—may be resolved by choosing a non-linear weighting with respect to fitness. A simple approach to this is *rank selection*: the population is placed in ascending order according to fitness, then an individual's selection probability is weighted according to their position in the ordering. Another approach is *tournament selection*, in which selection occurs by choosing pairs uniformly at random, then keeping the best of each pair for selection. Weights may also be assigned by first normalising fitnesses using a sigmoid function, thus exponentiating the absolute differences.

Further modifications to selection can be made. In *Boltzmann selection*, selection pressure begins low, corresponding to near-uniform weights, then gradually increases, corresponding to greater and greater weighting towards fitter individuals. The idea is that in earlier generations, diversity should be high to avoid premature convergence, then in later generations we can be more certain that the fittest individuals do indeed correspond to the global rather than some local maximum, allowing us to close in on the best solution faster. Employing *elitism*, we always carry the s_{elites} best chromosomes unchanged into the next generation. This ensures that we never go "backwards", losing the best solution we have seen so through crossover and mutation. s_{elites} should be kept low relative to s in order to maintain diversity and avoid premature convergence.

Crossover

Crossover emulates the process of reproduction between parents. It is typical to use two parents, emulating nature, although it is possible to use more. For simplicity, we will focus on the two-parent case and refer to them as the mother and father. In a typical, single-point crossover, we choose a locus l at random, that is a position in

the chromosome or equivalently an index of the bit-vector representing it. The first child's chromosome will copy the mother's chromosome to the left of the locus, and the father's to the right. The second child's chromosome takes on the reverse: the father's to the left, the mother's to the right.

Multipoint crossover, where two or more loci are chosen, can be more effective. In the two-point case, the first child would take on the mother's chromosome outside the chosen loci, and the father's chromosome inside. The second child would take on the inverse. Higher-point crossovers proceed similarly.

For each set of parents chosen, crossover occurs with a specified probability p_{cross}, typically set ≥ 0.75. If it does not occur, the parents' chromosomes are taken unchanged into the next generation, modulo mutation.

Mutation

Assuming a bit-vector chromosome, mutation is carried out by flipping each bit in the chromosome with probability $\frac{1}{l}$ where l is the length of the chromosome. Mutation occurs with a specified probability p_{mut}, which is typically set very low, ≤ 0.01. Mutation helps to maintain diversity in the population, and can be helpful in escaping local maxima.

7.3 Parameters

Above, we have identified the following parameters which must be chosen before running a GA:

- s: the population size.
- s_{elites}: the number of best individuals to take unchanged into the next generation, if using elitism.
- p_{cross}: the probability of performing crossover on a pair of selected parents.
- p_{mut}: the probability of performing mutation on a newly generated child.

There is no one-size-fits-all choice for these parameters, although empirical studies have been carried out in an effort to arrive at well performing parameters settings that can be used as a baseline. Schaffer et al. spent over a year of CPU time testing combinations of parameters, arriving at a recommended population size of 20-30, crossover rate of 0.75-0.95, and mutation rate of 0.005-0.01 [26]. Zeroing in on the best combination for a particular problem will of course be a matter of trial and error. Population size in particular may vary significantly. In part due to differences in computational

cost of evaluating the fitness function, but also, there is a trade-off between population size and number of generations iterated in a given time-frame, which will vary between problems.

7.4 The Building Block Hypothesis

Efforts to explain the success of genetic algorithms remain largely speculative—finding a rigorous theoretical basis for their performance remains an open topic of research. The *Building Block Hypothesis* (BBH), which can be traced back to the work of Holland [12], is a widely circulated conjecture in this direction. While the BBH is subject to criticism and should not be construed as providing a mathematically rigorous foundation for GA's, it nonetheless offers useful intuition as to why they *might* be very successful in searching a very large solution-space.

Roughly, the BBH suggests that the best solutions to a problem are made up of a number of *building blocks*. A GA works by finding and combining such building blocks, over time tending towards a better solution. This idea is formalised in terms of *schemata*. A schema identifies a set of bit-vectors. For example, the schema (1****0) is representative of all bit-vectors of length 6 that begin with a 1 and end in a 0. Ones and zeros in a schema are the *defined* bits, while asterisks are wild-cards or "don't care" bits. A bit-vector is an *instance* of a schema if it matches the schema at all of its defined bits.

Any bit-vector of length l is an instance of 2^l different schemata. The first key assertion of the BBH is that when a GA evaluates the fitness of a single chromosome, it is implicitly estimating the *average fitness* for a much larger region of the fitness landscape, corresponding to every schema of which it is an instance. This is referred to by Holland as *implicit parallelism*. The second is that the schemata which are implicitly evaluated form the building blocks of solutions, the best of which are being kept and further tested as the GA moves through generations. Furthermore, with each generation, the estimate of the fitnesses of the fittest schemata become more and more accurate as a larger sample of their instances is tested.

There are some *very* strong assumptions at play in the BBH. The obvious one is that schemata form the building blocks of solutions, and that combining good ones can be expected to produce better ones. It certainly cannot be stated that this holds in general, and whether or not it actually holds for a particular GA for a particular problem is heavily dependent on the problem, and the choice of how its solutions are encoded as chromosomes. The deeper underlying assumption is that there is some *structure* in the fitness landscape to be exploited. In the most extreme case, imagine

a fitness function that is arrived at by a random mapping of bit-vectors to numerical values. It would be completely unreasonable to expect in this scenario that combining a pair of high fitness chromosomes through crossover and mutation would produce a fitter one—it could not be expected to do any better than simply choosing another chromosome at random. More generally, it cannot be expected without further qualification that "nearby" chromosomes in the fitness landscape will have similar fitnesses, and that combining two good chromosomes will take us to an even better place in the fitness landscape. Again, whether or not there is structure in the fitness landscape, and whether that structure is exploitable by combining schemata, is heavily problem and encoding dependent. Burjorjee's paper on the topic provides a robust criticism of the building block hypothesis [4].

7.5 When to use a Genetic Algorithm

Genetic algorithms are suited to problems where the solution-space is far too large to search using complete methods, while the fitness landscape is too complex or "rugged" to traverse using other techniques. If the fitness landscape is smooth with only one or a few peaks, hill climbing or gradient ascent methods are likely to perform better. If the fitness landscape is convex or can be well approximated by a convex function, traditional optimisation methods are likely to be a better option. Furthermore, GA's should be used in domains where imperfect solutions can be tolerated, as a typical GA cannot provide guarantees on solution quality. On the other hand, the assumptions of the BBH regarding structure in the fitness landscape need to be taken into account before deciding to use a GA. If the fitness landscape is chaotic with no discernible structure, or if there is no way to encode solutions to the problem such that combining good ones may reasonably be expected to produce better ones, then there is no reason to believe that a GA will perform well. Finally, because a GA relies on a significant number of evaluations of the fitness function, it should be relatively inexpensive to compute.

Chapter 8

A Genetic Algorithm for MRMP

We know that if we want to find the best solution to an MRMP instance, in general, at least some coupling has to occur. But we also know that the size of the search-space grows exponentially with coupling. With this trade-off in mind, how might we go about finding the highest quality solution we can while keeping coupling low? This question, combined with the analogy between dynamic coupling and the phenomena of gradual complexification in real complex systems seen in Chapter 6 motivates an evolutionary approach. This chapter introduces the ConstraintGA: a genetic algorithm for MRMP where the chromosome represents a combination of constraints.

8.1 ConstraintGA

The genetic algorithm presented here is the coalescence of a number of ideas from the previous chapters. Starting from the most fundamental, we have seen that dynamic coupling allows us to maintain theoretical guarantees of completeness and optimality while potentially only searching a small subset of the explosively-sized configuration-space. We have seen that dynamic coupling is closely analogous to the phenomenon of gradual complexification that occurs in real-world complex systems, and that this phenomenon makes systems more evolvable—more likely to evolve towards the highest peak in the fitness landscape. Treating the number of collisions as a quantity to be minimised allows us to frame MRMP as such a complex system. Employing relative entropy, we were able to formalise a fitness function that is maximised by low-collision, low-coupling solutions. We have also seen that the implicit coupling induced

by building up a set of constraints, as in CBS, results in a lower average degree of coupling and thus notably improved practical performance, compared to the more direct, course-grained coupling that occurs in M* through sub-dimensional expansion.

We are now gestured towards a genetic algorithm for a number of reasons. Firstly, biological evolution is perhaps the original and prototypical example of a gradually complexifying system. It becomes immediately compelling to consider an algorithm motivated by its processes for MRMP, given the analogy with dynamic coupling. Furthermore, a genetic algorithm will allow us to leverage two properties of natural complexifying systems that Malkin and Lotto identified as desirable, but that existing dynamically coupled algorithms do not: randomness and occasional decomplexification. Finally, building up a set of constraints clearly matches the assumption of the building block hypothesis that fit solutions can be combined to build fitter ones: if constraint a resolves a collision at position p and time t, and constraint b resolves a collision at position q and time u, it is reasonable to expect that using both constraints might resolve both collisions. Of course, this will not always be the case: resolving one collision might create another—this essentially encapsulates what makes MRMP difficult: if all it took was identifying the collisions and resolving them in one pass, it would be an easy problem. But the validity of the assumption that combining constraints can lead to a solution is explicitly demonstrated in the behaviour and success of CBS.

8.1.1 Encoding

Each gene in the chromosome of the ConstraintGA represents a unique constraint, as defined in CBS. That is, we have vertex constraints of the form (r, p, t), enforcing that robot r cannot move onto position p at time t, and edge constraints of the form (r, p_{src}, p_{dst}, t), enforcing that robot r cannot move onto position p_{dst} *from* position p_{src} at time t. Abstractly, this chromosome can be thought of as a bit-vector where each bit indicates whether or not a particular constraint is active. In practice, it is useful to structure the chromosome as a 5-dimensional array, with dimensions $n \times m_w \times m_h \times d \times 5$. d corresponds to the time dimension, which could in theory have arbitrary size. In practice, however, it has been sufficient to set $d = 2(m_w + m_h)$, i.e. twice the maximum possible distance between two points on the grid. Indexing a chromosome H at $H[r, x, y, t]$ yields a bit-vector of length 5, where each bit corresponds to a constraint for the indexed robot r, position (x, y), and time t. The first bit corresponds to the clash constraint, bits 2 to 5 correspond to overlap constraints with p_{src} to the north, south, east, and west of (x, y), respectively. Every unique chromosome then corresponds to a plan $P(H)$ arrived at by finding a shortest path for every robot from start to target while obeying their corresponding constraints active in H.

8.1.2 Fitness Function

A slight modification of the fitness function (6.4) defined in Chapter 6 is used:

$$f(H) = D(p_{P(H)}) \cdot \sigma\left(\frac{H_{\max_k} - H_k}{H_{\max_k}}\right) \tag{8.1}$$

$p_{P(H)}$ is the collision probability for the plan $P(H)$, and D relative entropy, as defined in Section 6.3.1. H_k is the number of constraints active in H, and $H_{\max_k}$ is the maximum number of constraints possible (i.e. the product of the dimensions of H).

The main difference from (6.5) is that the input to the sigmoid function is the number of activate constraints rather than the degree of coupling: the amount of coupling is implicit in the number of constraints, so it is this that we will (de)complexify on. The τ term to penalise high objective values is discarded: because $P(H)$ is arrived at by finding a shortest constraint-obeying path, the resulting plan is naturally very low cost. In theory, H could have unnecessary constraints active that do not resolve any collisions but divert a robot and increase the plan's cost, however in practice it was found that the resulting plans rarely had suboptimal makespan, while keeping the τ term only added noise to the fitness function.

Complexity

The cost of evaluating the fitness of each generation dominates the complexity of any genetic algorithm. Evaluating f involves a run of A* for each robot, giving a worst case run-time in $O(n \cdot m_w m_h)$. Then the worst-case run-time of evaluating each generation is then in $O(s \cdot n \cdot m_w m_h)$ where s is the population size. The ConstraintGA is highly parallelisable: per generation, the fitnesses can be evaluated in parallel, and within the evaluation of each fitness the run of A* for each robot can be executed in parallel.

8.1.3 Operators

The encoding and fitness function given above are enough to plug into the general GA framework. Some problem-specific decisions on how the genetic operators are carried out were made, which are detailed here.

Initial Population

The stock method of creating an initial population is to take a uniform sample from the space of possible chromosomes. However, relative to the vast number of possible combinations of constraints, very few of them will be useful in resolving collisions. The likely result of this approach in the ConstraintGA is that no individuals in the

initial population resolve any collisions, and the algorithm quickly converges on a very shallow local maximum based on which member of the population has the least constraints active. Instead, the collisions in the naive solution to the instance are identified, and corresponding constraints created. Then, individuals for the initial population are chosen by sampling from the pool of possible k_{init}-sized combinations of the those constraints. k_{init} is a parameter that can be thought of as the "initial complexity". To allow for gradual complexification it should be low, but it must be greater than 1 else there will not be enough combinations to create an adequately diverse initial population. $\log n$ was found to be a good value for k_{init}.

Selection

Stochastic universal sampling is used for the selection procedure, with weights directly determined by fitness. Elitism is employed, taking the best few individuals into the next generation unchanged to ensure that the fitness of the population is non-decreasing across generations.

Crossover

Two-point crossover with two parents is used for reproduction. As an additional step, during crossover there is a small probability of activating a non-active constraint, with probabilities weighted by distance of the constraint from a collision in the plan $P(H)$ where H is the parent that constraint is being inherited from. Probabilities are set such that the expected number of constraints activated through this process is 1. The reason for this as that the space of constraints is vast, but looking at the collisions occurring in parent's solutions makes it possible to identify those that are most likely to be useful. This way, we are not dependent entirely on mutation to bring new constraints into the population. Not every choice in the design of a genetic algorithm need be motivated by biology, but if pressed, we could say that this process is an analogy with intra and intergenerational learning and adaptation.

Mutation

Mutation occurs as standard by flipping the value of bits of chromosomes selected for mutation. It was found to be useful to mutate with slightly greater "magnitude" than usual: the probability of any given bit being flipped was set such that the expected number of flips in a mutated chromosome was logarithmic in the size of the chromosome, rather than 1.

8.2 Experimental Evaluation

Experiments were run in an effort to answer the question set RQ2 listed in Chapter 2, repeated here.

RQ2:
1. With a fitness function designed such that low-collision solutions are the fittest, how effective is a genetic algorithm in resolving collisions?
2. How does the solution-quality of the ConstraintGA starting in a simple state then gradually complexifying compare to when it begins in a complex state?
3. Is the performance of dynamic-fully-coupled algorithms improved by starting with initial solutions with fewer collisions, provided by pre-processing with the ConstraintGA?

8.2.1 Setup

The ConstraintGA was tested on 8×8-grid instances with n ranging from 4 to 16, running either for 1024 generations or until a collision-free solution was found. For each value of n, there were 5 instances generated for each cluster-factor of 0, 0.33, 0.66, and 1, resulting in 20 instances per n, and 260 total instances. The coupling behaviour was compared at a fine-grained level on a few specific instances to a variant of the ConstraintGA modified not to prioritise gradual complexification. The solutions from the ConstraintGA were also provided as initial solutions to CBS to determine if it improved performance.

The GA parameters used were $s = 32$, $s_{\text{elites}} = 2$, $p_{\text{cross}} = 0.95$, and $p_{\text{mut}} = 0.01$. These parameters were arrived at simply by trial-and-error, using the recommendations from the work of Schaffer et al. [26] as a starting point.

8.2.2 Hypotheses

If the analogies explored in Chapter 6 have practical weight, it is expected that the ConstraintGA will be effective in minimising collisions (RQ2.1). For the same reasons, it is expected that the algorithm will have favourable performance when starting in a simple state and complexifying compared to when starting in a complex state (RQ2.2). In Chapter 5 we saw that there is a strong correlation between the run-time of CBS and M* and the number of collisions in the initial, naive solutions provided to them. Therefore it is expected that providing initial solutions with reduced collisions will improve performance (RQ2.3).

8.2.3 Results

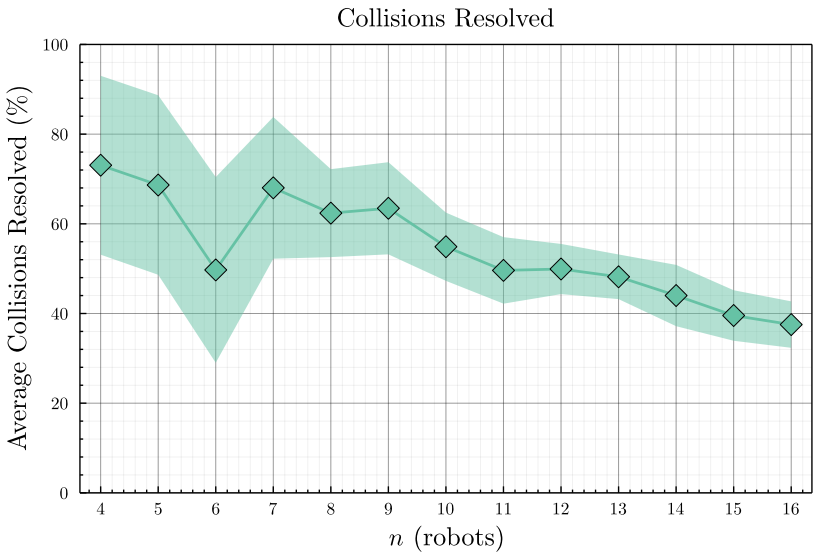
The ConstraintGA was fairly consistently able to resolve more than 50% of collisions, although this reduced gradually as n increased. This is illustrated in Figure 8.1.a.

Figure 8.2 illustrates the progress of the ConstraintGA on an 8 robot instance with (a) and without (b) gradual complexification. Starting in a complex state, the algorithm made very quick initial gains, but then got stuck in a local maximum. It can also be seen in (b) that complexity fluctuates without much order. Starting in a simple state and prioritising low-coupling solutions resulted in the algorithm making steady progress towards a higher peak. There also appears to be order in the fluctuations in complexity visible in (a): after the initial gradual increase, improvements in the best solution are met with a steep increase in complexity. Then, as time passes without finding a new best solution, the algorithm gradually decomplexifies. This may be seen as consolidating present gains, finding the simplest way to the present best, putting the system back into a more evolvable state to find further improvements.

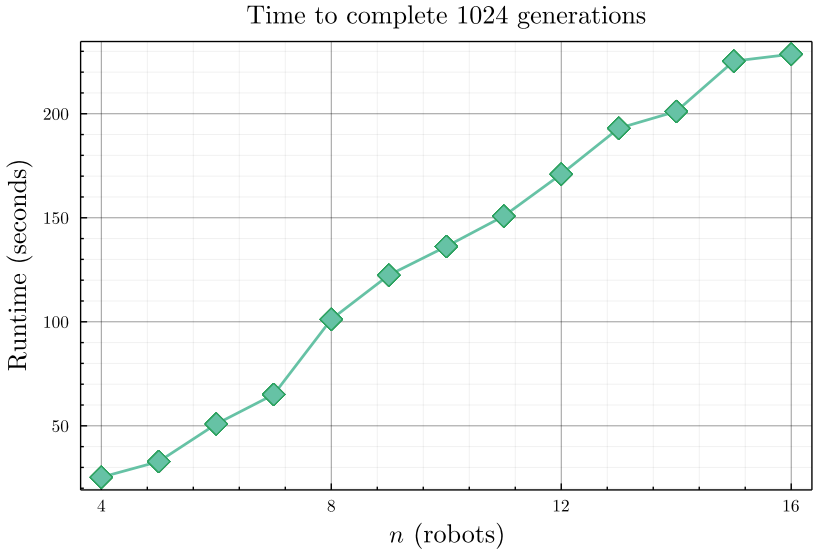
Providing preprocessed solutions to CBS did not see any notable improvements in performance: the success rate was either equivalent or only slightly higher compared to those seen in Chapter 5. This contradicted the initial hypothesis, based on the close relationship between run-time and collisions in the initial solution seen in Chapter 5. An explanation for this might be found in recalling the nature of complex systems. The interactions between components have a non-linear impact on fitness. In resolving *some* collisions, those interactions have changed. With respect just to the initial solution and the new preprocessed solution, it appears that we have made an improvement. But in the process we may have unwittingly created new adverse interactions which won't manifest themselves until a complete algorithm such as CBS takes on the heavy burden of bringing *complete* order to the system. When a naive solution to one instance has more collisions than a naive solution to another, this may be an indication that the first is fundamentally more complex than the latter. Even with a "partial", collision-reduced solution for the complex instance, it is still that same complex instance, and the complete algorithm still needs to find a way to handle that.

8.2.4 Conclusions

From a practical perspective, the results of this analysis could be seen as underwhelming. Solutions that still have ≈50% of the initial collisions don't have much real-world value, and using the ConstraintGA as a preprocessing step for CBS proved unfruitful. It was, however, validating of the theory behind this part of this thesis. Framing MRMP as a complex system with a fitness function that seeks to minimise entropy, so construed in terms of collisions, *can* trend towards feasible, optimal solutions. The

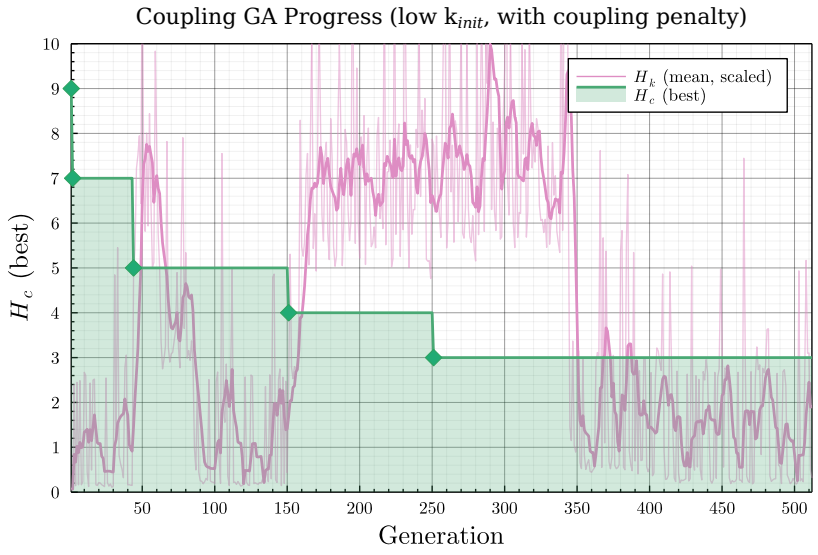


(a)

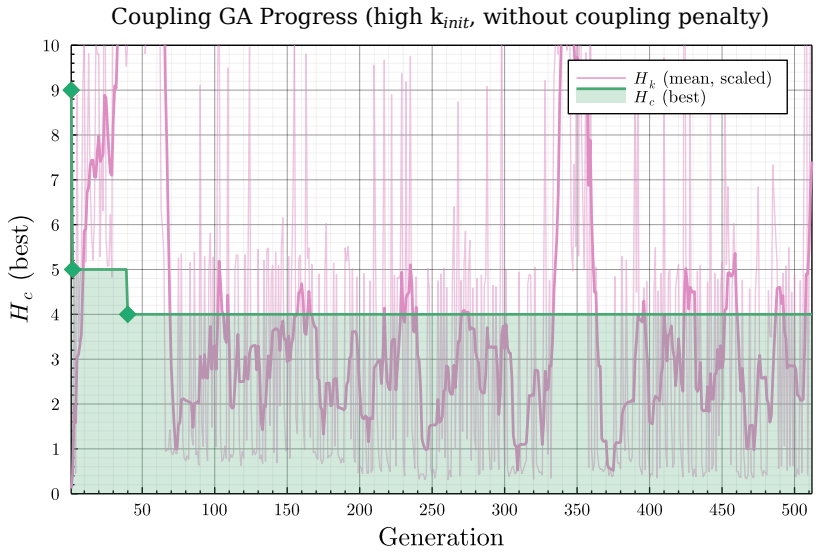


(b)

Figure 8.1: Percentage of collisions resolved by (a) and run-time of (b) the ConstraintGA. The ribbon in (a) indicates one standard-deviation.



(a)



(b)

Figure 8.2: Visualisation of the progress of the ConstraintGA running on an 8 robot instance with (a) and without (b) gradual complexification. H_c is the number of collisions in the plan corresponding to the best chromosome in a given generation. H_k is the number of constraints active in H: this value is averaged over all chromosomes in the given generation, and is scaled down to fit on the plot, so it is the *shape* of this line that should be emphasised.

analogy between dynamic coupling and gradual complexification proved to be beyond superficial, as the algorithm was indeed more evolvable when it embodied this behaviour, and demonstrated the viability of the strategy of maximising the solution-quality/coupling trade-off. The fields of complex systems and nature inspired computation are vast, and these results have provided enough to indicate that there is much to be gained for MRMP by exploring them further, equipped with the ideas presented here.

Conclusion

This thesis has taken a deep dive into the role that coupling plays in the Multi-Robot Motion Planning problem (MRMP). Clearly, the need to avoid collisions is what makes MRMP hard. By finding an algorithm-independent measure of coupling (Definition 3.1.3), it was possible to show that in order to avoid all collisions, all the time, and to assure optimality, coupling *must* occur (Theorems 4.2.1 and 4.3.1). But an increase in coupling corresponds to an increase in the dimensionality of the observed configuration-space, resulting in a potentially much harder search. These theoretical results solidify the intuitively expected trade-off between coupled and decoupled approaches to MRMP. Dynamically coupled algorithms emerged from this theory as a clear candidate for effectively managing this trade-off, and their success was empirically verified in Chapter 5.

MRMP has all the indicators of a complex system: finding a solution involves managing the interactions between many components, and the way those interactions influence the fitness of a solution is non-linear, perhaps chaotic. Indeed, if all it took to resolve collisions was to identify what collides and where, this would be an easy problem. But resolving one collision might cause another. Viewing MRMP through the lens of complex systems revealed two captivating analogies: (i) the technique of dynamic coupling closely parallels the phenomenon of gradual complexification that is manifested in naturally occurring complex systems, and is known to make systems more evolvable; and (ii) minimising collisions in an MRMP system can be viewed as minimising entropy, or bringing order to a chaotic system. (i) and (ii) combined to motivate a fitness function for MRMP (Equation 6.5) that allows it to be framed as a complex system that is highly evolvable by taking advantage of gradual complexification. These ideas were consolidated in a genetic algorithm, which showed promising ability to minimise collisions. Empirical evaluation of the genetic algorithm provided validation of the concepts that motivated it. It must be conceded that the genetic algorithm likely has limited practical use as is, since it typically returns solutions that still have some collisions. But if this work has shown one thing, it is that a system

without conflict might be more of a Platonic ideal than something to expect from one that is adequately complex.

Future Directions

Much work on MRMP already operates under the assumption that completeness and optimality cannot be guaranteed efficiently, and so focuses on heuristics. In light of the results in Part 2 of this thesis, it is nonetheless interesting to see how far dynamic coupling can be stretched. The success of CBS has shown that fine-grained approaches to coupling can keep the average actual search-space small while maintaining completeness and optimality. The notions of local-coupling and time-local-coupling discussed in Chapter 5 may offer starting points in this direction.

That MRMP is NP-hard raises the question of how the notions of collisions and coupling pertain to NP problems in general. Do they embody something fundamental that is in some way at play in all NP problems? The reduction to MRMP from Monotone 3-SAT shown by Demaine et al. [7] offers a starting point for addressing this question. Other NP problems involving notions of conflict, such as colouring problems, are also compelling candidates.

Complex systems, evolutionary programming, and nature inspired computation are vast fields of which only the surface has been scratched here. The findings in Part 3 of this thesis have provided enough to suggest that there may be practical and theoretical gains to be had by further exploring the relationship between MRMP and complex systems, and using this relationship to motivate algorithmic approaches.

Bibliography

[1] Manuel Abellanas et al. “Moving coins”. In: *Computational Geometry* 34.1 (Apr. 2006), pp. 35–48.

[2] Sergey Bereg, Adrian Dumitrescu, and János Pach. “Sliding Disks in the Plane”. In: *International Journal of Computational Geometry & Applications* 18.05 (Oct. 2008), pp. 373–387.

[3] J.P. van den Berg and M.H. Overmars. “Prioritized motion planning for multiple robots”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 430–435.

[4] Keki Burjorjee. “The Fundamental Problem with the Building Block Hypothesis”. In: *ArXiv* (Oct. 2008).

[5] Soon-Jo Chung et al. “A Survey on Aerial Swarm Robotics”. In: *IEEE Transactions on Robotics* 34.4 (Aug. 2018), pp. 837–855.

[6] Loïc Crombez et al. “Shadoks approach to low-makespan coordinated motion planning”. In: *Leibniz International Proceedings in Informatics, LIPIcs*. Vol. 189. 2021.

[7] Erik D. Demaine et al. “Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch”. In: *SIAM Journal on Computing* 48.6 (2019), pp. 1727–1762.

[8] Vishnu R. Desaraju and Jonathan P. How. “Decentralized path planning for multi-agent teams with complex constraints”. In: *Autonomous Robots* 32.4 (May 2012), pp. 385–403.

[9] Adrian Dumitrescu and Minghui Jiang. “On reconfiguration of disks in the plane and related problems”. In: *Computational Geometry* 46.3 (Apr. 2013), pp. 191–202.

[10] Sándor P Fekete et al. “Computing Coordinated Motion Plans for Robot Swarms: The CG:SHOP Challenge 2021”. In: *ArXiv* ().

[11] Sándor P. Fekete et al. “Methods for Improving the Flow of Traffic”. In: *Organic Computing — A Paradigm Shift for Complex Systems*. Basel: Springer Basel, 2011, pp. 447–460.

[12] John Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1972, p. 211.

[13] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. “On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman’s Problem"”. In: *The International Journal of Robotics Research* 3.4 (Dec. 1984), pp. 76–88.

[14] Ruoyun Huang, Yixin Chen, and Weixiong Zhang. “A Novel Transition Based Encoding Scheme for Planning as Satisfiability”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1 (2010).

[15] Wm. Woolsey Johnson and William E. Story. “Notes on the "15" Puzzle”. In: *American Journal of Mathematics* 2.4 (Dec. 1879), p. 397.

[16] Henry Kautz and Bart Selman. “Unifying SAT-based and Graph-based Planning”. In: *Proceedings of the 16th International Joint Conference on Artifical Intelligence* 1 (1996), pp. 318–325.

[17] Paul Liu et al. *Coordinated Motion Planning Through Randomized k-Opt*. 2021.

[18] Beau Lotto. *Deviate: The Science of Seeing Differently*. London: Orion Publishing Group, 2017, p. 256.

[19] David Malkin. “The Evolutionary Impact of Gradual Complexification on Complex Systems”. PhD thesis. University College London, 2009, p. 233.

[20] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Cambridge: MIT Press, 1999, p. 158.

[21] G. Ramanathan and V. Alagar. “Algorithmic motion planning in robotics: Coordinated motion of several disks amidst polygonal obstacles”. In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. Institute of Electrical and Electronics Engineers, pp. 514–522.

[22] Daniel Ratner and Manfred Warmuth. “The (n
2-1)-puzzle and related relocation problems”. In: *Journal of Symbolic Computation* 10.2 (Aug. 1990), pp. 111–137.

[23] Ralf Regele and Paul Levi. “Cooperative Multi-Robot Path Planning by Heuristic Priority Adjustment”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2006, pp. 5954–5959.

[24] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm". In: *Science* 345.6198 (Aug. 2014), pp. 795–799.

[25] Oren Salzman, Michael Hemmer, and Dan Halperin. "On the Power of Manifold Samples in Exploring Configuration Spaces and the Dimensionality of Narrow Passages". In: *IEEE Transactions on Automation Science and Engineering* 12.2 (Apr. 2015), pp. 529–538.

[26] J D Schaffer et al. "A study of control parameters affecting online performance of genetic algorithms for function optimization". In: *Proceedings of the third international conference on Genetic algorithms* January (1989), pp. 51–60.

[27] Jacob T. Schwartz and Micha Sharir. "On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers". In: *The International Journal of Robotics Research* 2.3 (Sept. 1983), pp. 46–75.

[28] Guni Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (Feb. 2015), pp. 40–66.

[29] Kiril Solovey, Oren Salzman, and Dan Halperin. "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning". In: *The International Journal of Robotics Research* 35.5 (Apr. 2016), pp. 501–513.

[30] Roni Stern et al. *Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*. Tech. rep. 2019.

[31] Glenn Wagner and Howie Choset. "Subdimensional expansion for multirobot path planning". In: *Artificial Intelligence* 219 (Feb. 2015), pp. 1–24.

[32] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI Magazine* 29.1 (2008), pp. 9–19.

[33] Hyeyun Yang and Antoine Vigneron. "A Simulated Annealing Approach to Coordinated Motion Planning". In: *37th International Symposium on Computational Geometry (SoCG 2021)* 189 (2021).

[34] Jingjin Yu. "Average case constant factor time and distance optimal multi-robot path planning in well-connected environments". In: *Autonomous Robots* 44.3-4 (2020), pp. 469–483.

[35] Jingjin Yu and Steven M Lavalle. “Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics”. In: *IEEE Transactions on Robotics* 32.5 (2016), pp. 1163–1177.

[36] Jingjin Yu and Daniela Rus. “Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms”. In: *WAFR*. 2015, pp. 729–746.