Solving multi-agent path planning by local search algorithms

Wang, Wenjie

2014

Wang, W. (2014). Solving multi-agent path planning by local search algorithms. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/62106>

<https://doi.org/10.32657/10356/62106>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**Solving Multi-Agent Path Planning by Local Search
Algorithms**

Wang Wenjie

School of Computer Engineering

A Thesis Submitted to the Nanyang Technological University
in Fulfillment of the Requirement for the Degree of
Doctor of Philosophy

2014

Acknowledgement

First of all, I sincerely thank my supervisor Associate Professor Goh Wooi Boon. As my supervisor, he is always very nice and patient in guiding me in my research. As an experienced mentor of mine, he is very enthusiastic in sharing his experience and skills with me. During the past five years, I indeed learnt a lot from him. I really appreciate his help.

Secondly, I want to thank some of my friends like Shimin Gong, Shaohua Li and Pham Manh Tung, etc for their valuable suggestions in my research work. Then I want to thank Nanyang Technological University for giving the opportunity to pursue my P.H.D study and providing all the facilities and resources I need.

Finally, I want to thank my parents and my sisters for their support and their encouragement towards my research work.

Table of Contents

Abstract.....	v
List of Figures.....	vii
List of Tables	xii
1. INTRODUCTION.....	1
1.1 Motivation.....	3
1.2 Objectives.....	4
1.3 Problem Definition	5
1.3.1. Makespan Minimized MAPP Problem	9
1.3.2. Total Cost Minimized MAPP Problem	10
1.4 Problem Formulation and Challenges	11
1.5 Contributions in the Thesis	13
1.6 The Thesis Organization.....	14
2. LITERATURE REVIEW.....	16
2.1 Coupled Approaches for MAPP	16
2.2 Decoupled Approaches for MAPP.....	20
2.3 Algorithms for MAPP Optimization Problems	24
3 ITERATIVE PATH COORDINATION FOR MAPP	27
3.1 Mathematical Background of Optimal MAPP Problems.....	27
3.1.1 Constrained Optimization.....	28
3.1.2 Discrete Optimization.....	32
3.2 Iterative Path Coordination for MAPP.....	34
3.2.1 Cost function, Local Neighborhood and Navigation Strategy	35
3.2.2 Parameter Setting Strategies for Penalty Coefficient μ	38
3.3 Solving MAPP Optimization Problems using IPC.....	42
3.3.1 Iterative Path Coordination for Makespan Minimized MAPP	42
3.3.2 Iterative Path Coordination for Total Cost Minimized MAPP	46
3.4 Experimental Evaluation.....	49

3.4.1	Evaluation of IPC-based Algorithms with Increasing Allowable Computational Time	50
3.5	Summary.....	58
4	ITERATIVE PATH COORDINATION USING GUIDED SEARCH.....	59
4.1	Guided Search Techniques.....	59
4.2	Guided Iterative Path Coordination (GIPC).....	61
4.3	Solving MAPP Optimization Problems using GIPC.....	64
4.3.1	Guided Iterative Makespan Minimization for Makespan Minimized MAPP.....	64
4.3.2	Guided Iterative Cumulation Minimization for Total Cost Minimized MAPP	66
4.4	Experimental Evaluation.....	67
4.4.1	Comparative Study of GIPC-based Algorithms under Varying Agent Count	67
4.4.2	Evaluation of GIPC-based Algorithms' Computational Efficiency under Different Environment Characteristics.....	72
4.5	Summary.....	78
5	ITERATIVE PATH COORDINATION USING STOCHASTIC SEARCH	80
5.1	Stochastic Search Techniques	80
5.2	Probabilistic Iterative Path Coordination (PIPC).....	82
5.2.1	Potential Function.....	83
5.2.2	Local Probabilistic Model	85
5.2.3	Parameter Tuning Strategies for Local Probabilistic Model.....	88
5.3	Solving MAPP Optimization Problems using PIPC	93
5.3.1	Probabilistic Iterative Makespan Minimization for Makespan Minimized MAPP....	93
5.3.2	Probabilistic Iterative Cumulation Minimization for Total Cost Minimized MAPP.....	94
5.4	Experimental Evaluation.....	96
5.4.1	Evaluation of the PIPC Algorithm	96
5.4.2	Comparative Evaluation of PIPC-based Algorithms under Varying Map Size	103
5.4.3	Comparative study of PIPC-based Algorithms under Varying Agent Count.....	105
5.5	Summary.....	111
6	ITERATIVE PATH COORDINATION USING HYBRID HEURISTIC	113
6.1	Reactive Iterative Path Coordination (RIPC)	113
6.1.1	Probabilistic Detection Mechanism.....	115

6.1.2	Switching Scheme	117
6.2	Solving MAPP Optimization Problems using RIPC	120
6.2.1	Reactive Iterative Makespan Minimization for Makespan Minimized MAPP	120
6.2.2	Reactive Iterative Cumulation Minimization for Total Cost Minimized MAPP	121
6.3	Experimental Evaluation.....	122
6.3.1	Evaluation of RIPC with Increasing Allowable Computational Time.....	122
6.3.2	Comparative Evaluation of RIPC with GIPC and PIPC	128
6.3.3	Comparative Study of RIPC-based Algorithms and Other Existing MAPP Algorithms	133
6.3.4	Comparative Evaluation of RIPC in Solving Challenging MAPP Scenarios	145
6.4	Summary.....	149
7	CONCLUSION AND FUTURE WORK	150
7.1	Summary of Contributions.....	150
7.2	Future Work.....	155
	PUBLISHED WORKS	159
	PUBLICATION UNDER REVIEW.....	159
	REFERENCES.....	160
	APPENDIX A1. Lemma 1 and Lemma 2.....	172
	APPENDIX A2. Proofs of Theorem 1 and Theorem 2	176

Abstract

Multi-Agent Path Planning (MAPP) in discrete space requires finding a collision-free path for each agent such that every agent can move from their starting positions to their respective destinations without colliding with other agents and static obstacles. With additional optimality criterion imposed, the MAPP problems can become intractable as the number of agents or the map size increases. This thesis specifically addresses two MAPP optimization problems, namely total energy cost minimized MAPP and makespan minimized MAPP. The total energy cost minimized MAPP aims at finding an optimal MAPP solution with respect to a cumulative objective function, while makespan minimized MAPP attempts to find an optimal MAPP solution with respect to a minimax objective function. Though finding an optimal solution for these two MAPP problems can be solved by existing optimal MAPP algorithms, the required computational time can be unacceptable even for relatively small map size and number of agents.

This work aims at finding a sufficiently good MAPP solution in a reasonable time and therefore adopts a local search approach for MAPP since local search algorithms are widely used to find a good enough solution for computationally hard problems in a reasonable time. A basic local search algorithm called Iterative Path Coordination (IPC) for MAPP is first introduced. The IPC algorithm iteratively improves the path quality for one of the agents with respect to a cost function until no further improvement is possible within the current MAPP solution or a preset time bound is reached. Based on this framework, three variants of IPC are proposed to address the inherent drawbacks of IPC, such as the problem of being trapped in local optima and the high computational cost. The first is a deterministic variant called Guided Iterative Path Coordination (GIPC) which adopts a guided cost function for MAPP.

Each time an infeasible locally optimal solution is encountered, the guided cost function is modified. In doing so, the guided cost function helps steer the search away from infeasible local optima. The second is a stochastic variant called Probabilistic Iterative Path Coordination (PIPC) that employs a probabilistic solution generation mechanism for MAPP. The mechanism generates a better neighboring solution with a higher probability. As such, an optimal MAPP solution can be quickly obtained with the highest probability. The final variant of IPC is a hybrid technique called Reactive Iterative Path Coordination (RIPC). This hybrid approach is able to combine the strengths of both the deterministic and stochastic variants as it has PIPC's computational efficiency in searching for a solution in the local neighborhood while maintaining GIPC's superior ability of escaping from infeasible local optima.

The performance of the proposed IPC-based algorithms were separately evaluated and compared with other MAPP algorithms designed for total cost minimized and makespan minimized MAPP. The comparative results show that the proposed variants of IPC strike a good tradeoff between the solution quality and the computational time incurred. With a limited computational time, they are able to outperform existing state-of-the-art MAPP algorithms in finding a superior MAPP solution using both the optimality criteria evaluated.

List of Figures

Figure 1-1. (a) A MAPP problem within a loop scenario and the viable paths (yellow arrows) required by agents $A1$ and $A2$ to get to their respective destinations $G1$ and $G2$; (b) The equivalent graphical representation of the MAPP problem within a loop scenario.	1
Figure 1-2. The graphical model of a MAPP problem within a tunnel scenario.	2
Figure 1-3. The graphical model of a MAPP problem within a tree scenario.	2
Figure 1-4. An interactive educational system for teaching children spelling. A swarm of cube-like mobile robots helps the child by rearranging themselves to form words (e.g. ‘HELLO’). An overhead camera observes the instantaneous location and orientation of each agent in the environment.	4
Figure 1-5. A MAPP solution with the lowest makespan for three agents in a 5x5 grid map. The solid circles marked with $A1$, $A2$ and $A3$ denotes mobile agents at their initial positions, while the dashed circles marked with $G1$, $G2$ and $G3$ are their respective destinations. The rectangles O denote static obstacles and the bold arrows show the paths of each mobile agent and its respective path cost.	9
Figure 1-6. A MAPP solution with the lowest total traversal distances for three agents in a 5x5 grid map. The solid circles marked with $A1$, $A2$ and $A3$ denotes mobile agents at their initial positions, while the dashed circles marked with $G1$, $G2$ and $G3$ are their respective destinations. The rectangles O denote static obstacles and the bold arrows show the paths of each mobile agent and its respective path cost.	11
Figure 3-1. Pseudocode for the Iterative Path Coordination (IPC) Algorithm for MAPP.	38
Figure 3-2. Overview description of the Iterative Makespan Minimization (IMM) algorithm for solving makespan minimized MAPP.	45
Figure 3-3. Overview description of the Iterative Cumulation Minimization (ICM) algorithm for solving total cost minimized MAPP.	47
Figure 3-4. A random scenario in a 30x20 grid map with 120 agents. Static obstacles occupying 20% of all accessible grid nodes are denoted by shadowed areas. The solid grey squares denote the mobile agents and the dashed squares denote their respective unique destinations.	49
Figure 3-5. Success rate of IMM with different μ setting strategies against allowable computational time.	51
Figure 3-6. Task completion time of IMM with different μ setting strategies against allowable computational time.	52
Figure 3-7. Success rate of IMM with different increment rates ε_c against allowable computational time.	53
Figure 3-8. Task completion time of IMM with different increment rates ε_c against	

allowable computational time.	54
Figure 3-9. Success rate of ICM with different cost ratios against allowable computational time.	55
Figure 3-10. Cumulative cost of ICM with different cost ratios against allowable computational time.	56
Figure 3-11. Task completion time of ICM with different cost ratios against allowable computational time.	56
Figure 4-1. Pseudocode for the Guided Iterative Path Coordination (GIPC) algorithm.	63
Figure 4-2. Overview description of the Guided Iterative Makespan Minimization (GIMM) algorithm for solving makespan minimized MAPP.	64
Figure 4-3. Overview description of the Guided Iterative Cumulation Minimization (GICM) algorithm for solving total cost minimized MAPP.	66
Figure 4-4. Comparative success rates of IMM and GIMM under varying agent count in a 30x20 grid map.	68
Figure 4-5. Comparative task completion times of IMM and GIMM under varying agent count in a 30x20 grid map.	68
Figure 4-6. Comparative computational times of IMM and GIMM under varying agent count in a 30x20 grid map.	69
Figure 4-7. Comparative success rates of ICM and GICM under varying agent count in a 30x20 grid map.	70
Figure 4-8. Comparative cumulative costs of ICM and GICM under varying agent count in a 30x20 grid map.	71
Figure 4-9. Comparative computational times of ICM and GICM under varying agent count in a 30x20 grid map.	71
Figure 4-10. Comparative success rates of GIMM with different static obstacle density in a 30x20 grid map.	74
Figure 4-11. Comparative success rates of GIMM with different ratios of the agent count to static obstacle density in a 30x20 grid map.	75
Figure 4-12. Comparative success rate of GIMM under varying map sizes against increasing iterations within GIPC.	76
Figure 4-13. Comparative computational time of GIMM under varying map sizes against increasing iterations within GIPC.	76
Figure 4-14. Comparative success rate of GIMM under varying map sizes against allowable computational time.	77

Figure 5-1. Probabilistic Path Planner in the Space-Time map.	87
Figure 5-2. Pseudocode for the Probabilistic Iterative Path Coordination (PIPC) algorithm for MAPP.	92
Figure 5-3. Overview description of the Probabilistic Iterative Makespan Minimization (PIMM) algorithm for solving makespan minimized MAPP.	93
Figure 5-4. Overview description of the Probabilistic Iterative Cumulation Minimization (PICM) algorithm for solving total cost minimized MAPP.	95
Figure 5-5. Success rate of PIMM with different β parameter setting strategies against the allowable computational time.	98
Figure 5-6. Task completion time of PIMM with different β parameter setting strategies against the allowable computational time.	99
Figure 5-7. Success rate of PIMM with different η parameter setting strategies against the allowable computational time.	101
Figure 5-8. Task completion time of PIMM with different η parameter setting strategies against the allowable computational time.	101
Figure 5-9. Comparative success rate of PIMM and GIMM with different map sizes against increasing iterations.	103
Figure 5-10. Comparative computational time of PIMM and GIMM with different map sizes against increasing iterations.	104
Figure 5-11. Comparative task completion time of PIMM and GIMM with different map sizes against increasing iterations.	105
Figure 5-12. A random scenario in a 4x4 grid map with 15 agents. The solid grey squares denote the mobile agents and the dashed squares denote their respective unique destinations.	106
Figure 5-13. Comparative success rates of GIMM and PIMM under varying agent count in a 4x4 grid map.	107
Figure 5-14. Comparative task completion time of GIMM and PIMM under varying agent count in a 4x4 grid map.	107
Figure 5-15. Comparative computational time of GIMM and PIMM under varying agent count in a 4x4 grid map.	108
Figure 5-16. Comparative success rate of GICM and PICM under varying agent count in a 4x4 grid map.	110
Figure 5-17. Comparative cumulative cost of GICM and PICM under varying agent count in a 4x4 grid map.	110
Figure 5-18. Comparative computational time of GICM and PICM under varying agent count in a 4x4 grid map.	111

Figure 6-1. Pseudocode for the Reactive Iterative Path Coordination (RIPC) algorithm for MAPP.	119
Figure 6-2. Overview description of the Reactive Iterative Makespan Minimization (RIMM) algorithm for solving makespan minimized MAPP.	120
Figure 6-3. Overview description of the Reactive Iterative Cumulation Minimization (RICM) algorithm for solving total cost minimized MAPP.	121
Figure 6-4. Success rate of RIMM with different possibility values p against allowable computational time in a 30x20 grid map (moderately crowded environment).	123
Figure 6-5. Task completion time of RIMM with different possibility values p against allowable computational time in a 30x20 grid map (moderately crowded environment).	124
Figure 6-6. Success rate of RIMM with different possibility values p against allowable computational time in a 4x4 grid map (highly crowded environment).	124
Figure 6-7. Task completion time of RIMM with different possibility values p against allowable computational time in a 4x4 grid map (highly crowded environment).	125
Figure 6-8. Success rate of RIMM with different sample sizes π against allowable computational time in a 4x4 grid map.	127
Figure 6-9. Task completion time of RIMM with different sample sizes π against allowable computational time in a 4x4 grid map.	127
Figure 6-10. Comparative success rates of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.	129
Figure 6-11. Comparative task completion times of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.	129
Figure 6-12. Comparative computational times of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.	130
Figure 6-13. Comparative success rates of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.	130
Figure 6-14. Comparative task completion times of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.	131
Figure 6-15. Comparative computational times of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.	131
Figure 6-16. Comparative success rate of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.	135
Figure 6-17. Comparative task completion time of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.	136

Figure 6-18. Comparative computational time of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.	136
Figure 6-19. Comparative success rate of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.	137
Figure 6-20. Comparative task completion time of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.	137
Figure 6-21. Comparative computational time of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.	138
Figure 6-22. Comparative success rate of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.	140
Figure 6-23. Comparative cumulative cost of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.	141
Figure 6-24. Comparative computational time of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.	141
Figure 6-25. Comparative success rate of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.	142
Figure 6-26. Comparative cumulative cost of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.	142
Figure 6-27. Comparative computational time of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.	143
Figure 6-28. Comparative task completion times of the P&S and RICM algorithms under varying agent count in a 30x20 grid map.	144
Figure 6-29. The three challenging MAPP problem evaluated, namely (a) Loop, (b) Tunnel and (c) Tree scenarios. The top row shows the simulation environment used for the experiment and the bottom row shows the equivalent graphical model of each of the respective MAPP scenarios.	145
Figure 7-1. Overview description of the iterative algorithm for solving makespan minimized MAPP.	153

List of Tables

Table 1-1. The taxonomy of the various MAPP algorithms proposed in this thesis.	14
Table 6-1. Comparative results of the MAPP algorithms in solving challenging MAPP scenarios. For successful scenarios, their computational times (rounded to the nearest millisecond) are given in brackets.	146

1. INTRODUCTION

Multi-agent path planning (MAPP) is currently an active research topic in autonomous agents and artificial intelligence. In single agent path planning, given a starting point and a target point in an environment with static and moving obstacles, the path planner is required to determine an optimal path between these two points based on some associated cost functions related to the path and the motion sequence of the agent. The research for single agent path planning is mature and lots of algorithms have been proposed. In the more challenging multi-agent domain, however, each agent is often required to move from its starting point to its destination simultaneously without colliding with each other or static obstacles. Finding feasible MAPP solutions is often a non-trivial problem.

Figure 1-1. (a) A MAPP problem within a loop scenario and the viable paths (yellow arrows) required by agents A1 and A2 to get to their respective destinations G1 and G2; **(b)** The equivalent graphical representation of the MAPP problem within a loop scenario.

Take for example the MAPP problem within a loop scenario illustrated in Figure 1-1(a). Two agents A1 and A2 need to reach their respective positions G1 and G2. However, the path of one agent is blocked by the other if they both move along their respective shortest path. To resolve this situation, one agent has to give way to the other and cycles along the entire loop to reach its destination. As such, priority assignments should be given to the agents in order

to find viable paths for both of them. Figure 1-1(b) shows the graphical representation of the same loop scenario.

Figures 1-2 and 1-3 illustrate the graphical models of some other challenging MAPP scenarios. In Figure 1-2, two agents A1 and A2 are moving concurrently to their destinations in a tunnel scenario. In this scenario, no matter which one is assigned a higher priority, the other one will fail to reach its destination. This is because the high priority agent will first reach the middle node and therefore block the way out of the low priority agent. The only solution is to slide one of them to the node marked with dotted grid and then keep it there temporarily so as to give way to the other one. In Figure 1-3, the agent A1 is already at its destination while the agents A2 and A3 need to swap their positions in a tree scenario. To allow agents A2 and A3 to swap their positions, the agent A1 needs to move away from the junction and temporarily stay at the top node while agents A2 and A3 execute maneuvers similar to the tunnel scenario in Figure 1-2.

Figure 1-2. The graphical model of a MAPP problem within a tunnel scenario.

Figure 1-3. The graphical model of a MAPP problem within a tree scenario.

1.1 Motivation

The MAPP problems in discrete space arise in many real life applications, such as computer game, automatic warehouse management, vehicle routing problem and so forth. In computer game like Warcraft, different human characters are often required to reach different destinations without collision in a 2D map so as to complete some target tasks. In the automatic warehouse management problem, goods are delivered by a group of automatic mobile robots from some pickup points to different shelves for storage. In vehicle routing problem, a number of customers are required to be served by a fleet of vehicles. In all these applications, the whole environment and the position of each agent are often observable, so planning the viable paths or routes for each agent can be done in a centralized manner. Apart from finding viable paths, some additional goals are often imposed. For example, a minimal total route cost is desired in vehicle routing problem. The specific MAPP goals addressed in this work were motivated by the application scenario in Figure 1-4, which shows a tangible interactive system for teaching children how to spell words in the English language. This system consists of a set of passive and active cubes marked with letters of the English alphabets, which the child can arrange to form words. The active cubes are intelligent autonomous mobile robots that can interact and assist the child by maneuvering themselves to appropriate localities so as to provide meaningful contextual scaffold to the child while he is forming a word. Overseeing the entire workspace is a video camera that is able to track the position and orientation of each individual letter block and mobile robot, essentially providing an instantaneous map of the entire operating environment. The central host PC uses this map to plan the traversal path of relevant mobile robots so that they can quickly travel to appropriate positions in the map to form a required word.

Figure 1-4. An interactive educational system for teaching children spelling. A swarm of cube-like mobile robots helps the child by rearranging themselves to form words (e.g. ‘HELLO’). An overhead camera observes the instantaneous location and orientation of each agent in the environment.

1.2 Objectives

In order for the mobile robots or agents in the interactive system in Figure 1-4 to move to their respective destinations without colliding with each other or static obstacles, finding viable paths for each agent is the first and primary objective of this work. To be responsive, the mobile agents have to quickly form the required letter sequence for the child once the target word formulation is known. As such, mobile agents are required to move to their respective positions as soon as possible. As a result, one of the goals required in the path planning is to minimize the makespan. This is the time duration from the initiation of agent movement until the time the last agent reaches its desired position.

As the mobile agents are battery operated, it is therefore desirable that the task can be completed in the most energy efficient manner possible. Another goal addressed in this work is to find the paths for these agents with the lowest total energy consumptions, which is incurred by the movement of all agents to reach their desired positions. In this thesis, the makespan and energy minimization goals are treated as two separate MAPP optimization

problems. This is because these two MAPP minimization problems do not often share the same optimal solution, which is discussed in detail in section 1.3.1 and 1.3.2. Since the interactive system has to react to the child with the minimal latency, computing these qualified paths using the designed offline MAPP algorithms should also be done within a reasonable time.

In summary, the objectives of this work are to design computationally fast MAPP algorithms that are able to find sufficiently good paths for each agent based on the makespan and total energy cost minimized criteria.

1.3 Problem Definition

The MAPP problem in discrete space for n agents is commonly described as the problem of finding a feasible path r_i for each agent A_i ($i=1,2,\dots,n$) in a graph-based environment such as visibility graph [1], Voronoi diagram [2], grid map [3] and so on, so that each agent can move from a starting point $S_i=(x_{s_i}, y_{s_i})$ to its unique destination $G_i=(x_{G_i}, y_{G_i})$ without colliding with other agents or static obstacles O .

In this work, time is discretized into a series of unit time steps. For simplicity, the environment is modeled on a four connected undirected grid map and one of the five unit actions (i.e. North, South, East, West, Idle) is allowed to be chosen for each agent at each unit time step. Choosing one of the four possible moving actions traverses the agent from one node to the neighboring node in the four-connected grid map, while choosing the idle action makes the agent stay at its current node. The assumption that all mobile agents are of unit size allows the starting and destination locations of each agent to be directly mapped to the vertices in the grid map. However, this work can be applied to general abstract graphs only on the assumption that each unit action takes a uniform time interval. If this assumption is

violated, additional synchronization and approximation strategies will be used to ensure collision-free path planning for each agent. Assume the position of each agent A_i for $i \in \{1, \dots, n\}$ at the time step t is denoted by the space-time path node $r_i(t) = (x, y, t)$, then the space-time path r_i of each agent A_i with the total time step T_i is denoted by a sequence of path nodes $(r_i(0), \dots, r_i(t), \dots, r_i(T_i))$ where $r_i(0) = (x_{s_i}, y_{s_i}, 0)$ and $r_i(T_i) = (x_{G_i}, y_{G_i}, T_i)$. Once an agent reaches its destination, adding further idle actions does not change its trajectory in space and time. As a result, these paths with the same trajectory in space and time but different total time step T_i are defined as the equivalent paths.

Definition 1. Equivalent paths – Two different paths r_i^a and r_i^b for $i = 1, 2, \dots, n$, are equivalent if $r_i^a(t) = r_i^b(t)$ for $t \in \{1, \dots, \min(T_i^a, T_i^b)\}$ and $r_i^a(t) = (x_{G_i}, y_{G_i}, t)$ if $T_i^a > T_i^b$, or else $r_i^b(t) = (x_{G_i}, y_{G_i}, t)$ for $t \in \{\min(T_i^a, T_i^b) + 1, \dots, \max(T_i^a, T_i^b)\}$.

Definition 2. Equivalent solutions – Two different solutions r^a and r^b are equivalent if their respective components r_i^a and r_i^b for $i = 1, 2, \dots, n$, are the same or equivalent.

Since all agents are allowed to move concurrently in an environment with static obstacles, two types of collisions arise. The first is *node collision*, which happens when two agents A_i and A_j come to the same node at the same unit time step; for example when

$$r_i(t) = r_j(t) \text{ for each } \{i, j, t\} \quad (1a)$$

where $i \in \{1, \dots, n\}$, $j \in \{1, \dots, n\}$, $j \neq i$, $t \in \{1, \dots, \max(T_i, T_j)\}$, or one agent A_i moves to a node occupied by static obstacles such as

$$r_i(t) = L(O_k) \text{ for each } \{i, k, t\} \quad (1b)$$

CHAPTER 1. INTRODUCTION

where $i \in \{1, \dots, n\}$, $k \in \{1, \dots, m\}$, $t \in \{1, \dots, T_i\}$ and $L(O_k)$ denotes the position of the static obstacle O_k . The other is *head-on collision*, which happens when the two agents A_i and A_j move concurrently along the same edge from opposite directions; which can be formally described by

$$\{ r_i(t)=r_j(t-1) \text{ and } r_i(t-1)=r_j(t) \} \text{ for each } \{i, j, t\} \quad (1c)$$

where $i \in \{1, \dots, n\}$, $j \in \{1, \dots, n\}$, $j \neq i$, $t \in \{1, \dots, \max(T_i, T_j)\}$.

Assume the functions $\Omega_{i,j}(t)$, $\varphi_i(t)$, $\phi_{i,j}(t-1, t)$ indicate the value (1 if true, or else 0) of the Boolean expression in (1a), (1b) and (1c) respectively. In order to guarantee a feasible solution, the following conditions are required to be satisfied:

$$\varphi_i(t)=0 \text{ for each } \{i, t\} \quad (2a)$$

where $i \in \{1, \dots, n\}$, $t \in \{1, \dots, T_i\}$.

$$\Omega_{i,j}(t)=0 \text{ and } \phi_{i,j}(t-1, t)=0 \text{ for each } \{i, j, t\} \quad (2b)$$

where $i \in \{1, \dots, n\}$, $j \in \{1, \dots, n\}$, $j \neq i$, $t \in \{1, \dots, T_{i,j}\}$ ($T_{i,j} = \max(T_i, T_j)$). The constraints in (2a) and (2b) are divided into two types, namely *static constraint* and *dynamic constraint* respectively. In feasible problem instances, removing static obstacles does not destroy the connectedness of the grid map. As a result, the static obstacles are strictly satisfied, in other words, only the traversable space that is not occupied by static obstacles is explored. When $T_i < T_j$, the collision detection between agents can be done between r_j and the equivalent path r'_i of r_i with $T'_i = T_j$.

Definition 3. *Static constraint* - is defined as a collision constraint imposed by the static obstacles in function (2a) and it is not time related.

CHAPTER 1. INTRODUCTION

Definition 4. *Dynamic constraint* - is defined as a collision constraint imposed by the incremental paths of agents in function (2b) at any given time step.

Based on the constraints, the path nodes $r_i(t) = (x, y, t)$ for $t = 0, 1, \dots, T_i$ are divided into two different types of nodes, namely free path node v , collision path node $\bar{v}$.

Definition 5. *Free path node v* – is defined as a path node that satisfies both the collision constraints given in functions (2a) and (2b), and can therefore be freely traversed by an agent.

Definition 6. *Collision path node $\bar{v}$* – is defined as a path node that satisfies only the collision constraints given in function (2a) and may be traversed by an agent under some conditions.

Since each path consists of a sequence of path nodes, then the feasible path and feasible solution can be defined as follows:

Definition 7. *Feasible path r_i^f* – is a path consisting of only free nodes.

Definition 8. *Feasible solution $r^f = (r_1, \dots, r_n)$* – is defined as a solution in which each path r_i for $i=1, 2, \dots, n$ in r is a feasible path.

Definition 9. *Feasible region R^f* – is defined as the set of all possible solutions that satisfy the constraints (2a) and (2b).

The simplest MAPP problem is to find a feasible solution, as described at the beginning of this section. A more challenging MAPP problem is to find an optimal solution with respect to an additional optimality criterion. This work mainly addresses two common MAPP optimization problems, namely makespan minimized MAPP and total cost minimized MAPP.

1.3.1. Makespan Minimized MAPP Problem

The makespan minimized MAPP problem is defined as: starting from different initial positions, all agents are required to move to their unique destinations. The task is deemed completed when the last agent reaches its destination and the goal is to minimize this task completion time. Assume the function $\Phi_i(r)$ for $i \in \{1, \dots, n\}$ measures the total time related cost of all unit actions in the path r_i , then the makespan minimized MAPP problem is mathematically described as

$$\min_r F_m(r) = \max_{1 \leq i \leq n} \Phi_i(r) \quad (3)$$

Subject to (2a) and (2b). As uniform time is used for five unit actions in this work, so the makespan minimized MAPP problem is equivalent to finding a MAPP solution with the minimal number of unit actions for the last agent.

Figure 1-5. A MAPP solution with the lowest makespan for three agents in a 5x5 grid map. The solid circles marked with $A1$, $A2$ and $A3$ denotes mobile agents at their initial positions, while the dashed circles marked with $G1$, $G2$ and $G3$ are their respective destinations. The rectangles O denote static obstacles and the bold arrows show the paths of each mobile agent and its respective path cost.

An example of the makespan minimized MAPP problem is shown in Figure 1-5. At each unit time, each agent can either traverse to the neighboring grid node or stay idle at its current grid node. The optimal makespan MAPP solution results in a maximum of 8 unit times, which is incurred by the last agent $A2$.

1.3.2. Total Cost Minimized MAPP Problem

The total cost minimized MAPP problem is defined as: starting from different initial positions, all agents are required to move to their unique destinations in the shortest possible total traversing cost. In this work, the total cost minimized MAPP problem attempts to find an energy efficient MAPP solution, which has the minimal total energy consumption. Since the energy consumption is often incurred by movements, so here the energy consumption of each agent is assumed to be proportional to the distance moved. In other words, idle unit action has zero energy cost. Assume the function $\Psi_i(r)$ for $i \in \{1, \dots, n\}$ measures the total distances traversed by the agent A_i , then the total cost minimized MAPP with respect to energy consumption can be mathematically described as

$$\min_r F_c(r) = \sum_{i=1}^n \Psi_i(r) \quad (4)$$

Subject to (2a) and (2b). As such, the total energy cost minimized MAPP is solved by finding an optimal MAPP solution with the minimal total distances traversed. Figure 1-5 has shown an optimal makespan MAPP solution; however, the optimal makespan MAPP solution incurs a total of 15 traversal unit distances. Figure 1-6 gives an energy efficient MAPP solution for the same scenario in Figure 1-5, as it only incurs a total of 13 traversal unit distances. On the contrary, this energy efficient solution is not makespan optimized as it leads to a maximum of 9 unit times incurred by agent $A3$.

Figure 1-6. A MAPP solution with the lowest total traversal distances for three agents in a 5x5 grid map. The solid circles marked with $A1$, $A2$ and $A3$ denotes mobile agents at their initial positions, while the dashed circles marked with $G1$, $G2$ and $G3$ are their respective destinations. The rectangles O denote static obstacles and the bold arrows show the paths of each mobile agent and its respective path cost.

1.4 Problem Formulation and Challenges

The MAPP problem in the graph-based environment can be formulated as a path planning problem in a composite state space. As defined in [4], “each distinct situation for the world is called a *state*, and the set of all possible states is called a *state space*.” In the state space for MAPP, the state s could represent the location and orientation of each agent in the search map. Since rotation is not considered in this work, so a state s is the set of location of each agent in the search map. In this case, the initial state s_I denotes the starting position of each agent while the goal state s_G denotes the destination position of each agent. The traversing from a state s to its adjacent s' is realized by executing one of allowable actions for each agent. To satisfy the additional collision constraints in the MAPP problem, the state with more than one agent occupying the same location in the search map is declared as *illegal*

state, otherwise, it is *legal state*. As such, multi-agent path planning is solved by searching the state space, starting from the initial state s_I , exploring and expanding the adjacent legal states until the goal state s_G is reached. As a result, the goal of multi-agent path planning is to find a route or path that connects a sequence of adjacent states with the initial state in the head and the goal state in the tail. Since the path in the composite state space provides each agent with a collision-free path that traverses the agent from its starting position to its destination without collisions, the path in the state space is also called a *solution* to the MAPP problem.

Since the state space for MAPP is a Cartesian product of the configuration space of each agent, the size of the state space increases rapidly with the number of agent and the number of vertices in the search map. As a result, it can become computationally expensive to find a target solution in this large state space, especially if an optimal solution is desired. The generalized MAPP problem, namely motion planning of multiple agents, has already been proved to be PSPACE-hard [5], [6] (A class of problems that are at least as hard as the hardest problems in PSPACE. A problem is PSPACE-hard if every problem in PSPACE can be polynomial time reducible to it. A PSPACE-hard problem is definitely NP-hard). However, some MAPP problems can be solved in polynomial time. Kornhauser et.al [7] provided a theoretical work that is able to solve the pebble motion problem on graphs in polynomial time and proved that an $O(n^3)$ moves are required for solutions. This is to say that non-optimal MAPP can also be solved efficiently. However, finding an optimal solution for MAPP is still a challenging task and is proved to be NP-complete [8].

For solving NP-complete [9] problems in an efficient manner, there are mainly three types of approaches. Though NP-complete implies none of complete polynomial time algorithm exists, it does not mean all the problem instances cannot be solved in polynomial time. As a result,

one type of the approach for NP-complete problem is designed to identify and solve all the problem instances that are solvable in polynomial time. The second type of this approach is the approximation algorithm, which is able to find an approximate solution with bounded solution quality for NP-hard problems. The last is the heuristic algorithm adopted in this thesis, which is able to find a good enough solution within a reasonable computational time.

1.5 Contributions in the Thesis

Various efficient heuristic algorithms were developed in this work to find a sufficiently good solution for both makespan minimized and total cost minimized MAPP problems. The summary of all proposed algorithms is given in Table 1.1. The major contributions of this thesis are summarized as follows:

- Proposed an iterative framework to approximate the optimal solution of the two specific MAPP optimization problems, namely the makespan minimized and total cost minimized MAPP, by solving a sequence of standard MAPP minimization problems.
- Proposed a local search algorithm named Iterative Path Coordination (IPC) for MAPP that is able to find a good MAPP solution based on the two specific MAPP objectives.
- Proposed a deterministic variant of IPC named Guided Iterative Path Coordination (GIPC), which applies guided local search for MAPP so as to help the search escape from infeasible local optima.
- Proposed a stochastic variant of IPC named Probabilistic Iterative Path Coordination (PIPC), which incorporates a problem-specific probabilistic model designed for MAPP such that a locally optimal MAPP solution can be quickly obtained with the highest probability.

- Proposed a hybrid variant of IPC named Reactive Iterative Path Coordination (RIPC), which integrates the GIPC's guided technique and PIPC's probabilistic solution generation mechanism for MAPP such that RIPC can combine each of their respective strengths.

Table 1-1. The taxonomy of the various MAPP algorithms proposed in this thesis.

	Iterative algorithms for makespan minimized (MM) MAPP	Iterative algorithms for total cost minimized (CM) MAPP	Local search algorithms for solving standard MAPP minimization problem
IPC-based algorithms	IMM (Details in Section 3.3.1)	ICM (Details in Section 3.3.2)	IPC (Details in Section 3.2)
GIPC-based algorithms	GIMM (Details in Section 4.3.1)	GICM (Details in Section 4.3.2)	GIPC (Details in Section 4.2)
PIPC-based algorithms	PIMM (Details in Section 5.3.1)	PICM (Details in Section 5.3.2)	PIPC (Details in Section 5.2)
RIPC-based algorithms	RIMM (Details in Section 6.2.1)	RICM (Details in Section 6.2.2)	RIPC (Details in Section 6.1)

1.6 The Thesis Organization

The thesis is organized into seven chapters and each of these is summarized here.

Chapter 2 surveys the related works for MAPP within the different categories of approaches adopted by past works.

Chapter 3 proposes a basic local search algorithm named Iterative Path Coordination (IPC) for MAPP and then provided the specific iterative algorithms for solving the makespan minimized and total cost minimized MAPP problems. The IPC-based algorithms were then evaluated under different performance criteria and parameter settings.

CHAPTER 1. INTRODUCTION

Chapter 4 proposes a novel guided local search algorithm called Guided Iterative Path Coordination (GIPC) for MAPP. In the experimental evaluations, the GIPC-based algorithms were compared with the IPC-based algorithms to demonstrate the improved performance in finding a feasible MAPP solution by adopting a guided technique.

Chapter 5 proposes a novel stochastic local search algorithm called Probabilistic Iterative Path Coordination (PIPC) for MAPP. Various parameter tuning strategies were proposed to improve its performance. The PIPC-based algorithms were evaluated with different parameter settings strategies and then compared with the algorithms proposed in chapter 4.

Chapter 6 proposes a hybrid local search algorithm named Reactive Iterative Path Coordination (RIPC) for MAPP. The RIPC-based algorithms were evaluated under different parameter settings and then compared with the GIPC-based, PIPC-based and other existing MAPP algorithms.

Chapter 7 concludes the work and discusses some directions for future work.

2. LITERATURE REVIEW

Multi-Agent Path Planning (MAPP) has been researched for many years. Finding a feasible solution in MAPP is a challenging task because the number of collisions between agents increases rapidly with the number of agents, leading to a lower probability of finding a feasible solution. In contrast with the MAPP feasibility problem, finding an optimal solution in MAPP is more challenging because it requires finding the best feasible solution with respect to some optimality criteria. The approaches for MAPP can be divided into two basic categories, namely coupled or decoupled approaches. Section 2.1 and section 2.2 will introduce various coupled and decoupled approaches for MAPP respectively. Section 2.3 will survey the related works for solving MAPP optimization problems.

2.1 Coupled Approaches for MAPP

Coupled approaches simultaneously plan the paths for all agents and perform search in a composite state space using graph search algorithms. The simplest graph search algorithms are the depth-first [10] and breadth-first [11] algorithms. The depth-first search algorithm keeps expanding one of the children nodes in each layer of a searching tree until the goal node is achieved or the end of a branch is reached. It terminates when the goal node is achieved. When it hits the end of a branch, it will backtrack one step and expand another branch of the backtracked node. By contrast, the breadth-first search algorithm is different from the depth-first search algorithm due to the way it expands children nodes. The breadth-first search algorithm first expands all children nodes in the same layer of the searching tree before it repeats the same procedure in the next layer. It terminates when the goal node is achieved or no unexpanded node is left. More computationally efficient algorithms like the Dijkstra [12] and A* algorithms [13] are able to reduce computational cost by eliminating

impossible paths through dynamic programming [14] or heuristics [15]. The Dijkstra algorithm is a lowest-cost first search algorithm. It repeatedly expands the current lowest cost node until the goal node is found. Let g be a cost associated with a node in the graph. It notes the exact path cost from the current node to the starting node. Besides, a data structure *openlist* is used to keep all explored nodes. The Dijkstra algorithm repeatedly expands the node with the least g value in *openlist* until the target node is found. Unlike the Dijkstra algorithm, A* repeatedly expands the current most promising node by heuristics. A* has three cost values g , h , and f . The cost g computes the path cost between the current node and the initial node. The cost h estimates the path cost between the current node and the target node (for example: Manhattan distance). And f is the sum of the cost g and h . The A* algorithm keeps a data structure called *openlist* for caching nodes to be expanded and another one called *closelist* for storing the expanded node. The nodes in *closelist* should not be explored and added back into *openlist* again. At the beginning, both *openlist* and *closelist* are empty and it puts the starting node into *openlist*. The starting node will then be expanded and the nodes around starting nodes will be explored and put into *openlist*. The expanded starting node will be deleted from *openlist* and put into the *closelist*. Then it will expand the next node with the least f value in *openlist*. The node in *openlist* will be replaced if the same node with a lower g value is added into *openlist*. This procedure is repeated until the target node is expanded, which in this case will cause the A* algorithm to return a successful search. If *openlist* eventually becomes empty, this signifies a failed search. Compared with Dijkstra, A* is often more computationally efficient in finding an optimal path or solution. However, finding an optimal path in a large state space is computationally expensive. In the MAPP problem, as the number of agent increases the computational time can become unacceptable. In this case, techniques like dynamic weight heuristic, hierarchical planning, bidirectional

search and beam search and so on can be used to make a tradeoff between the optimality and the computational complexity of the algorithm.

Dynamic weight sets a bigger weight to scale up the heuristic function at first so as to speed up the search at the beginning, and slowly reduces the weight to make sure the goal can be found. ARA*[16] is one such algorithm. At the beginning ARA* sets the weight ϵ to a value that is greater than 1 so as to find a solution rapidly. However, if the heuristic h is larger than the actual cost h^* from the current node to the target node, it does not guarantee an optimal path solution can be found. As a result, the ARA* algorithm can find a viable solution rapidly at the beginning, but the path solution is not optimal. ARA* tackles this problem by decreasing the weight ϵ and storing the expanded and refined nodes into a structure called ICONS in each planning step, and then uses the stored nodes and a smaller weight to search the path again. ARA* can return a sub-optimal solution at anytime but would be too time consuming if an optimal solution is required.

Hierarchical planning finds an abstract path solution in a map with a coarse resolution, and then refines it in a map with a higher resolution until the refining process cannot go down further or the optimal solution is found. The construction of a hierarchical map is very simple. Initially the search space is treated as a whole, and progressively split into disjoint spaces until a pre-determined resolution is achieved. Hierarchical planning saves computational time by searching in a reduced search space, but its solution is often sub-optimal.

Bidirectional search does the searches from the start point to the target point and from the target point to the start point in parallel. A final solution is obtained when these two searches in the opposite direction meet. This approach decomposes a big search tree into two small search trees. As a result, it reduces time complexity of the bidirectional A* algorithm.

Beam search limits the number of unexpanded nodes in A^* , and drops the worst node when a preset limit is exceeded. This approach can reduce memory cost, but it may drop the node which is likely to lead to the optimal solution in the worst case. An alternative approach is to set a limit to the search depth. Iterative deepening depth-first search (IDDFS) is a depth-limited search. It increases its depth limit iteratively until it reaches d , the depth of the shallowest goal state. At the beginning, the depth is set to 0. It increases by 1 at each planning cycle until the goal state is returned. In each planning cycle, IDDFS performs a depth-first search under the limit of depth. Based on IDDFS, IDA^* [17] uses f cost as the limit instead of the depth. At each iteration, IDA^* expands the node whose f cost is less than or equal to a threshold. The threshold is set to the estimated value of the initial state at the beginning, and then increases to the least f cost among unexpanded but explored nodes at each iteration. The advantage of IDA^* is it always remembers the current solution instead of the whole search tree.

Other techniques like rapidly exploring random tree (RRT) [18] and probabilistic roadmap [19] are helpful in reducing the size of search space. Svestka and Overmars [20] proposed a super graph to plan the paths for all agents. In this graph, each adjacent super node is reached by moving only one of the agents in a probabilistically generated roadmap. Unfortunately, their algorithm is only probabilistically complete due to their use of a probabilistic roadmap planner.

In summary, coupled approach can find an optimal solution. However, finding this optimal solution can become computationally intractable as the number of agents or the size of the map increases. Though many techniques have been proposed to speed up the search process, they inevitably sacrifice the completeness or optimality of the algorithms.

2.2 Decoupled Approaches for MAPP

Decoupled approach decomposes the MAPP problem into several sub-problems, and then solves each sub-problem separately. There are two types of decoupled approaches, namely prioritized planning and path coordination.

Prioritized planning based decoupled approach solves each sub-problem in a sequential manner, and the subsequent sub-problems are solved with respect to the previously obtained sub-solutions. Mors et al. [21] proposed a routing planning algorithm based on free time windows (FTW) to sequentially solve each single agent path planning sub-problems. The FTWs in each node are the time intervals during which the node is not occupied by other agents. Each agent finds its viable path sequentially by connecting the FTWs between the starting node and the destination. Since the higher priority agents do not consider the path planning process of the lower priority agents, they may inadvertently prevent lower priority agents from finding viable paths. Berg. et al. [22] proposed a constraint graph to find the independent groups of agents and their associated priority orders. However, in a highly crowded environment, all the components may be strongly connected in the constraint graph and this prohibits the decoupling of the problem into sub-problems. Wang and Goh [23] [24] proposed an adaptive priority re-assignment strategy to find a feasible priority order but this re-assignment can get trapped in a cyclical priority re-ordering loop. Bennewitz et al. [25] adopted a randomized and hill climbing search to find a near-optimal priority order. This approach starts with an arbitrary initial priority order and then iteratively changes the priorities of agents randomly so as to find a better priority order until some stopping criteria are satisfied. However, the fixed priority assignment among agents suffers from the priority racing problem in which each agent is competing for the top priority. Silver [26] proposed Windowed Hierarchical Cooperative A*(WHCA*) algorithm, which adopted a dynamic

priority assignment strategy. In this priority assignment strategy, each agent takes turn to have the top priority for a period of time. Since it is unknown how long each agent should be assigned the top priority each time, dynamic priority assignment is still a challenging task in WHCA*. Later Sturtevant and Buro [27] improved WHCA* with map abstraction, which helped reduce the memory and the initial initialization time. However, these improvements are obtained at the expense of the entire running time for finding a valid solution. In addition, they claim no improvement in path quality over the original WHCA*.

Path coordination based decoupled approach first finds an individual sub-solution for each sub-problem independently and then resolves the conflicts between the sub-solutions by using some coordination schemes. Thirty years ago, Röger and Helmert [28] described a theoretical work for finding a feasible MAPP solution in polynomial time. In recent years, many works had been done in an attempt to turn the theoretical work into a practical algorithm. Ryan [29] introduces a sub-graph decomposition method to partition the map into some special graph structures like stacks, cliques and singleton. The paths for all robots are firstly planned in the abstract map and then the paths in these sub-graphs are coordinated. He later proposed an extra hall sub-graph [30]. However, this algorithm suffers from a scalability issue because performing path planning for a large number of robots in the sub-graph based map can become computationally intractable due to his systematic search strategy. Wang and Botea [31] adopted a traffic control that allows one way movement in each row or column of the grid map and different movement directions for two adjacent rows or columns to construct a directed Flow-Annotated search graph. Then the path for each agent is found independently in the Flow-Annotated graph and the collisions between robots are resolved locally. However, pushing the blocking unit away from its target in order to give way to another unit may result in an endless cyclical swapping. They later proposed a different grid

map-based MAPP algorithm [32] that resolves the local collision between agents by temporarily pushing away lower priority agents into pre-computed alternative paths so as to accommodate high priority agents. This MAPP algorithm was further improved [33] by relaxing some assumptions so as to find viable paths for the agents which are not slidable. Though these algorithm extensions improved the number of solvable agents, their results did not demonstrate that all the agents are solvable even in a low map density with at most 2000 unit size mobile agents and at least 13765 traversable tiles. Peasgood et al. [34] proposed a decoupled approach for MAPP in a spanning tree. Their algorithm first moved all the agents to the leaf nodes in the tree and then adjusted or swapped the positions of agents such that the sequential paths which are based on the depth of their goals in the spanning tree, from deepest to the shallowest, are guaranteed to exist. However, their algorithm requires the strict condition that the number of the leaf is more than the number of agents and this does not always hold in a very crowded environment. Masehian and Nejad [35] proposed a linear time algorithm to check the solvability of a MAPP problem in a tree. Their algorithm partitioned the tree to different influence zones by the junctions, and the solution exists when every vertex belongs to an influence zone and each pair of influence zones is interconnected. Later Khorshid et al. [36] improved on their work and proposed a polynomial time algorithm to solve the MAPP problem in a tree. The agent moves to its goal by swapping positions with the agents along its path. The operation of swapping the positions of two adjacent agents is realized by using the nearest junction with enough holes in the solvable tree. Surynek [37] proposed the BIBOX algorithm for MAPP in bi-connected environments with at least two unoccupied vertices. He later proposed BIBOX- θ [38], which required only one unoccupied vertex in the bi-connected graph. Though the above algorithms are computationally efficient in finding a feasible solution for MAPP, they are highly dependent on the topology of the

search graph. Luna and Berkis [39] proposed a computationally efficient Push and Swap algorithm for a general class of MAPP problems that made no assumptions about the topology of the graph. Firstly, the Push and Swap algorithm moves the agents one by one to reach its destination along their pre-computed paths by the operation of pushing the lower priority agents away. When the higher priority agents block the way, another swap operation is triggered. In the swap operation, it swaps the position with the agent ahead and kept the positions of other agents unchanged. Sajid et al. [40] recently modified the Push and Swap algorithm into a parallel version that allows the concurrent movement of agents. Krontiris et. al [41] improved the computational efficiency of the Push and Swap algorithm by avoiding many redundant operations. The Push and Swap algorithm and its variants require at least two empty vertices in the graph. In other words, they will not be able to solve MAPP with one empty vertex in the graph like the well-known 15 sliding-tile puzzle [42]. Wilde claimed in [43] that the Push and Swap algorithm can still fail in some specific scenarios in which a feasible solution exists.

In summary, decoupled approaches perform solution search in lower dimensional space and is therefore computationally more efficient than coupled approaches. However, they often return sub-optimal solutions or do not guarantee a feasible solution can be found even when one exists. Though the computational efficiency of decoupled approaches can be improved by performing search in a lower dimensional search space, this often sacrifices the optimality and completeness of the algorithm.

2.3 Algorithms for MAPP Optimization Problems

Since the focus of this work is to solve the makespan minimized and total cost minimized MAPP problems, the related works for these two specific MAPP optimization problems will be reviewed in this section.

Makespan minimized MAPP: Yu and LaValle [44] proposed an optimization algorithm for solving makespan minimized MAPP. Their algorithm converted the multi-agent path planning problem into an equivalent multi-flow problem, and then solved a sequence of time-dependent integer linear programming (ILP) problems. Though this optimal MAPP algorithm is able to find an optimal solution, its number of flow edges increases rapidly with the number of agents and the size of the map. In a crowded environment that has a large number of agents, the computational time taken to return an optimal solution may become unacceptably long, which makes it inappropriate to be used in response to time-sensitive scenarios such as that shown in Figure 1-4.

Total cost minimized MAPP: Standley [46] proposed the Independence Detection (ID) algorithm that tries to find the independent groups of agents such that the entire problem can be solved by planning the paths for each independent group separately. In addition, he proposed a technique called A* [13] with Operator Decomposition (OD) that can lead to a significant reduction of node expansions during the path planning of each independent group if a perfect heuristic is available. Though the OD+ID algorithm can guarantee the optimality of the solution, perfect heuristic in MAPP is rare and independent groups are difficult to obtain in a crowded environment. In this case, the OD+ID algorithm may suffer from exponentially increasing computational time as the agent count increases. Standley and Korf [47] went on to modify OD+ID into a fast maximum group size (MGS) algorithm by dynamically dropping an optimality constraint and changing the expanding order criterion.

They then proposed the optimal anytime (OA) algorithm, an incremental version of MGS with gradually increasing group size. This algorithm can improve the solution quality if more time is allowed. Though dropping the two constraints in MGS and OA may help reduce the probability of group merging, it slows down the convergence rate to an optimal solution. In addition, the problem of group merging still cannot be adequately resolved. Wagner and Choset [48] introduced a sub-dimensional expansion strategy that dynamically generates the low dimensional search space and then proposed the M* algorithm based on it. Though the M* algorithm is able to find an optimal solution, it suffers from a rapidly increasing search space with the number of agents. Sharon et al.[49] proposed an increasing cost tree (ICT) to find an optimal solution with respect to a cumulative cost function. Each node in the ICT contains a vector of costs for the path of each agent, and the cumulative cost of each node in ICT increases with the depth of the tree. The node will be tested as a goal node if there is a feasible combination of each individual path with respect to its associated costs. Though ICT guarantees the globally optimal solution, the size of ICT grows exponentially with the depth of the optimal goal node. Sharon et al. [50] later proposed another optimal approach that solves the MAPP sub-problems sequentially according to a systematically increasing constraint tree. Though their approach is optimal and only a single agent path planning problem is required to be solved each time, the size of the constraint tree may increase rapidly with the number of possible collisions between agents.

Though the optimal MAPP algorithms for makespan minimized and total cost minimized MAPP presented are able to find an optimal solution, they inevitably suffer from rapidly increasing search space with the increasing agent count and map size. Since computational efficiency is an important consideration of this work, this thesis proposes various local search algorithms to find a sufficiently good MAPP solution in a reasonable time. Being a

decoupled approach, the proposed local search algorithms decompose the MAPP problem into several single-agent path planning sub-problems and then solve each sub-problem in an iterative manner. Since a single-agent path planning sub-problem is only addressed each time, so the proposed local search algorithms avoid the issue of rapidly increasing search space as the agent count increases. Moreover, being iterative algorithms, they take each search step by only considering the current solution and therefore address the problem of rapidly increasing memory requirements with the increasing agent count. To improve the performance of the proposed local search algorithms in solving some challenging MAPP optimization problems like makespan minimized MAPP, additional iterative frameworks were employed in this work. An earlier makespan minimized MAPP algorithm proposed by Wang and Goh [45] called Iterative Max-Min Improvement (IMMI) proposed an iterative framework for solving an equivalent inequality-constrained problem of makespan minimized MAPP. Their iterative framework decomposes the inequality-constrained problem into a linear maximization problem and a MAPP minimization problem and then approximates the optimal makespan MAPP solution by repetitively solving the maximization problem using Simplex method [70] and the MAPP minimization problem based on one of the proposed local search algorithms presented in this thesis until some stopping criteria are satisfied. Different from IMMI, the work presented in this thesis approximates the optimal makespan MAPP solution by employing an iterative framework for solving its smooth approximate problem. This new direction was adopted since the smooth approximate problem can be more easily and directly solved by available optimization tools or standard unconstrained optimization algorithms.

3 ITERATIVE PATH COORDINATION FOR MAPP

This chapter will first introduce the mathematical background of the MAPP optimization problem and then propose an iterative algorithm for solving the makespan minimized and total cost minimized MAPP problems.

3.1 Mathematical Background of Optimal MAPP Problems

The goal of an optimization problem is to find the element from a set of available alternatives that can yield the best measurable performance with respect to some given quantitative criteria. A general mathematical form of the optimization problem is described as

$$\min F(x) \tag{5}$$

$$\text{Subject to } g_i(x) \leq 0 \text{ and } h_j(x) = 0 \tag{6}$$

where $x \in A$ (A is some set), $i = \{1, \dots, m\}$, $j = \{1, \dots, n\}$, $F(x)$ is called the objective function that maps an element x to a real value in R , $g_i(x)$ and $h_j(x)$ are called inequality and equality constraint functions respectively. The goal of the optimization problem in (5) and (6) is to find the element x^* from the set A with the minimal value $F(x^*)$ while satisfying the constraints imposed in (6). The maximization problem can be converted to a minimization problem by negating the objective function.

Definition 10. The element x^* is called a locally optimal optimum if $F(x^*) \leq F(x')$ for any $x' \in N(x^*)$, where $N(x^*)$ is a subset of A and contains all neighboring elements of x^* .

Definition 11. The element x^* is called a globally optimal optimum if $F(x^*) \leq F(x')$ for all $x' \in A$.

There are many categories of optimization problems. According to the nature of the objective and constraint functions, they can be divided into smooth or non-smooth, linear or non-linear, convex or non-convex and so forth. According to the distinction between the problems with explicit constraints on variables or not, they can be divided into constrained or unconstrained optimization problems. According to the distinction that the variables take on real values or discrete values, they can be divided into continuous or discrete optimization problems.

The MAPP feasibility problem can be considered as an optimization problem without an objective function. In contrast, the total cost minimized MAPP problem is to find an optimal solution with respect to a cumulative cost function while makespan minimized MAPP tries to find an optimal solution with respect to a minimax function. In this work, they are all constrained, discrete optimization problems. The following sections will introduce approaches for addressing constrained and discrete optimization problems.

3.1.1 Constrained Optimization

The constrained optimization problem is to find the best solution from the set of feasible solutions that is determined by constraints. Its mathematical form has been shown in (5) and (6). To address constrained optimization problems, several types of approaches can be used. One type of the approach is the feasible direction method [51].

The *feasible direction methods* like Frank-Wolfe [52] and Zoutendijk [53] solve an optimization problem by moving from a feasible solution x_k to an improving feasible solution x_{k+1} along an improving feasible direction d_k . That is $x_{k+1} = x_k + \alpha * d_k$, where α is a positive step length such that x_{k+1} is a feasible solution and is better than x_k . Given the feasible solution x_k , the task here is to first find an improving feasible direction d_k according to the following conditions:

$$\nabla F(x_k)^* d_k < 0 \quad (7)$$

$$g_i(x_k + d_k) \leq 0, \quad h_j(x_k + d_k) = 0 \quad (8)$$

$$\|d_k\| \leq 1 \quad (9)$$

where $\nabla F(x)$ is the gradient of the objective function $F(x)$ over variable x . The condition (7) indicates that d_k is an improving direction, while the condition (8) implies that d_k is a feasible direction. The condition (9) is often called the normalization constraint. After the improving feasible direction d_k is obtained, it then finds an optimal step length α along the improving feasible direction d_k by solving a one-dimensional optimization problem given by

$$\min F(x_k + \alpha * d_k) \quad (10)$$

$$\text{Subject to} \quad g_i(x_k + \alpha * d_k) \leq 0, \quad h_j(x_k + \alpha * d_k) = 0 \quad (11)$$

$$\text{and} \quad \alpha \geq 0 \quad (12)$$

The feasible direction method can terminate when no improving feasible direction can be found.

An alternative approach for addressing the constrained optimization problems called *Lagrangian method* [54] tries to find the optimal solution by solving the necessary conditions for the optimality of the optimization problem. The Lagrangian method relaxes the constraints (6) by using Lagrange multipliers [55]. When only equality constraints $h_j(x)$ for $j = \{1, \dots, n\}$ exist, by using a Lagrange multiplier λ_j for each equality constraint $h_j(x)$, the Lagrange function is given by

$$L(x, \lambda) = F(x) + \sum_{j=1}^n \{\lambda_j * h_j(x)\} \quad (13)$$

Then the necessary condition for optimality of the Lagrange function in (13) is described as

$$\nabla L(x, \lambda)=0 \quad (14)$$

Or as partial derivatives given by

$$\nabla_x L(x, \lambda)=0 \text{ and } \nabla_\lambda L(x, \lambda)=0 \quad (15)$$

When inequality constraints $g_i(x)$ for $i=\{1, \dots, m\}$ are also considered, a non-negative Lagrange multiplier μ_i is used for each inequality constraint $g_i(x)$ and the Lagrange function is changed to

$$L(x, \mu, \lambda)=F(x)+\sum_{i=1}^m\{\mu_i * g_i(x)\}+\sum_{j=1}^n\{\lambda_j * h_j(x)\} \quad (16)$$

In this case, the optimality condition for the Lagrange function $L(x, \mu, \lambda)$ is given by

$$\nabla L(x, \mu, \lambda)=\nabla F(x)+\nabla \sum_{i=1}^m\{\mu_i * g_i(x)\}+\nabla \sum_{j=1}^n\{\lambda_j * h_j(x)\}=0 \quad (17)$$

Where x is called a primal variable while μ and λ are called dual variables. As $\nabla_\lambda L(x, \mu, \lambda)=0$ implies $h_j(x)=0$ for $j=\{1, \dots, n\}$, while $\nabla_\mu L(x, \mu, \lambda)=0$ implies $g_i(x)=0$ for $i=\{1, \dots, m\}$. However, $g_i(x)$ is an inequality constraint, so to satisfy the optimality condition in (17), $\mu_i^*=0$ if $g_i(x^*)<0$ at the optimal point (x^*, μ^*, λ^*) . In other words, $\mu_i^* * g_i(x^*)=0$ for $i=\{1, \dots, m\}$. As a result, an alternative expression of the optimality condition in (17) can be described by the following conditions

$$\nabla_x L(x^*, \mu^*, \lambda^*)=\nabla_x F(x^*)+\sum_{i=1}^m\{\mu_i^* * \nabla_x g_i(x^*)\}+\sum_{j=1}^n\{\lambda_j^* * \nabla_x h_j(x^*)\}=0 \quad (18)$$

$$g_i(x^*)\leq 0, h_j(x^*)=0, i=\{1, \dots, m\}, j=\{1, \dots, n\} \quad (19)$$

$$\mu_i^* \geq 0, i=\{1, \dots, m\} \quad (20)$$

$$\mu_i^* * g_i(x^*)=0, i=\{1, \dots, m\} \quad (21)$$

Where the condition (19), (20), (21) are often called the primal feasibility, dual feasibility and complementary slackness respectively. These optimality conditions (18)-(21) for the

Lagrange function $L(x, \mu, \lambda)$ are also called Karush-Kuhn-Tucker (KKT) condition [56]. The KKT condition can be solved by direct methods or iterative methods.

The third approach called penalty method [101] converts the constrained problem to an unconstrained problem by relaxing the constraints. The constraints are relaxed by adding a penalty term to the objective function. The penalty term measures the violation of the constraints. One of the common penalty methods called quadratic penalty method [57] uses a quadratic penalty function, in which the penalty term is the square of the constraint violations. The quadratic penalty function of the constrained optimization problem in (5) and (6) is given as

$$Q(x, \mu) = F(x) + \mu \sum_{i=1}^m ([g_i(x)]^-)^2 + \mu \sum_{j=1}^n (h_j(x))^2 \quad (22)$$

where μ is a non-negative penalty coefficient and $[g_i(x)]^-$ denotes $\max(g_i(x), 0)$. When the penalty coefficient μ is set to a sufficiently high value such that all the infeasible points x have a higher $Q(x, \mu)$ than feasible ones, in this case, the unconstrained minimization problem with (22) shares the same optimal point with the constrained minimization problem in (5) and (6). As a result, with a sufficiently high μ , minimizing the quadratic penalty function (22) is equivalent to solving the constrained minimization problem in (5) and (6). The optimal point of the function in (22) can be found directly by solving its necessary condition for optimality if the function $Q(x, \mu)$ is differentiable, or approximated by using iterative methods or unconstrained optimization algorithms. Beginning with an initial value of x_0 , the unconstrained optimization algorithms generate a sequence of iterates x_k ($k \in \{1, 2, \dots\}$) until no more improvement can be made or other specific termination conditions are satisfied. There are two main search strategies for unconstrained optimization algorithms, namely line search [58] and trusted region [59]. With the line search strategy, the

optimization algorithm first chooses a search direction d_k and then takes step along d_k from the current iterate x_k to search for an improving iterate x_{k+1} . With the trusted region strategy, the optimization algorithm first defines a region around the current iterate x_k by using an approximate model, which can be realized by using a quadratic model of the objective function, and then chooses the step to be the minimizer of the model in the defined region. When the objective and constraint functions are continuously differentiable, gradient method [60] and Newton's method [61] are both efficient algorithms to approximate an optimal solution. When they are non-differentiable, optimization algorithms like pattern search [62] are helpful.

3.1.2 Discrete Optimization

In general, discrete optimization includes integer programming [63] and combinatorial optimization [64]. Integer programming refers to the mathematical optimization problem in vector space. In integer programming problems, all or some of the variables are restricted to integer values. When all the variables are restricted to integer values, it is called a *pure integer programming* problem (PIP) and its mathematical form is given by

$$\max \sum_{i=1}^m \{c_i * x_i\} \quad (23)$$

$$\text{Subject to} \quad \sum_{i=1}^m \{a_{ji} * x_i\} \leq b_j, x_i \geq 0 \text{ and } x_i = 0, 1, 2, \dots \quad (24)$$

where $i=1, 2, \dots, m$ and $j=1, 2, \dots, n$. When the variables x_i for $i=1, 2, \dots, m$ are only allowed to be 0 or 1, it is called a *binary integer programming* problem (BIP), which is one of Karp's 21 NP-complete problems [65]. When some of the variables are restricted to integer values, it is called a *mixed integer programming* Problem (MIP). In contrast with integer programming, combinatorial optimization refers to the problems on discrete structures like graph, matroid and so on. The domain of the combinatorial optimization

problem is the set of discrete feasible solutions. Some examples of combinatorial optimization problems include the constraint satisfaction problem (CSP) [66], traveling salesman problem (TSP) [67], knapsack problem [68] and vehicle routing problem [69].

Integer programming is NP-hard. One common approach is to convert the integer programming problem to a linear programming problem by relaxing the integral constraints, and then approximate its optimal solution by linear programming methods like the Simplex method [70] or interior point method [71]. However, this approach does not guarantee an optimal solution or even a feasible solution can be found. When all the variables are restricted to integer values, the integer programming problems can be modeled as combinatorial optimization problems and be solved by combinatorial optimization methods.

The naïve method for combinatorial optimization is to enumerate all possible solutions. However, this method is practically intractable. Exact methods like branch and bound [72] and branch and cut [73] are more computationally efficient ways to find an optimal solution by pruning impossible solutions. When the problem is complex, exact methods like divide and conquer [74] and dynamic programming [14] are helpful in breaking down the complex problem into simpler sub-problems. As such, the overall solution can be obtained by solving each sub-problem.

Though exact methods are able to find an optimal solution, they may suffer from combinatorial explosion. Moreover, for the combinatorial optimization problems that are NP-hard, exact methods are not feasible. In this case, heuristic algorithms can be used to find an approximately good solution in a reasonable time. One of the popular heuristic algorithms is local search and it is adopted in this work. Beginning with an initial solution, the local search method iteratively replaces its current solution by a better neighboring solution until all candidate neighboring solutions are worse than the current solution. In this case, a locally

optimal solution is obtained. In a constrained problem or a multi-modal problem that contains multiple local optima, the returned locally optimal solution may be infeasible or inferior. Metaheuristics are heuristic algorithms that are designed to guide the low level procedure or heuristic such as local search to quickly find a sufficiently good solution. There are several categories of metaheuristic algorithms, including the single solution and population based algorithms. The single solution algorithms focus on improving a single candidate solution. They include techniques such as guided local search [75], iterated local search [76], simulated annealing [77], variable neighbor search [78], greedy randomized adaptive search procedure (GRASP) [79] and so forth. The population based algorithms simultaneously improve multiple candidate solutions. They include techniques such as genetic algorithm [80], genetic programming [81], memetic algorithm [82], scatter search [83], particle swarm optimization [84], ant colony algorithm [85], river formation dynamics [86], etc. This work in particular adopts the single solution metaheuristic algorithm because maintaining and improving multiple candidate MAPP solutions is relatively costly in terms of memory requirement, especially when the map size and agent count are large.

3.2 Iterative Path Coordination for MAPP

To address the MAPP optimization problems, this work adopts the local search method and proposes a novel local search algorithm for MAPP. The basic local search method iteratively moves from the current solution to an improving solution selected from a local neighborhood until no further improvement can be made or a time bound is reached. As a result, to apply the local search method for MAPP, three issues need to be addressed a priori, namely the cost function, local neighborhood and navigation strategy.

3.2.1 Cost function, Local Neighborhood and Navigation Strategy

Cost function indicates the solution quality and it can be derived from the objective function $F(r)$ in the target problem. However, in constrained optimization problems like those given in (3) and (4), additional constraints need to be satisfied. As such, at each iteration an improving neighboring solution is obtained with respect to the objective function $F(r)$ and the constraints (2a) and (2b). A common way is to combine them into one aggregate function. Since the static constraints (2a) are strictly satisfied, the dynamic constraints (2b) are relaxed by adding a penalty term to the objective function. In doing so, a general cost function for MAPP is given by

$$F_p(r; \mu) = F(r) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{\Gamma_{i,j}(r)\} \quad (25)$$

where $F(r)$ is a general objective function in a MAPP problem. μ is a non-negative penalty coefficient and it is desirable to be set to a high enough value for reasons discussed earlier in (22). $\Gamma_{i,j}(r)$ measures the violations of the dynamic constraints (2b) between the paths r_i and r_j and it is given by

$$\Gamma_{i,j}(r) = \sum_{t=1}^{T_{i,j}} \{\Omega_{i,j}(t) + \phi_{i,j}(t-1, t)\} \quad (26)$$

Local neighborhood includes candidate solutions. The choice of the local neighborhood has a significant influence on the performance of the local search method. A large local neighborhood may result in less number of locally optimal solutions, but may be computationally costly to evaluate if a best neighboring solution is desired. On the contrary, a small local neighborhood is more computationally efficient, but may produce more locally optimal solutions.

Definition 12. The move operator [97] specifies the allowable modifications to one component r_i of the current solution $r^{cur}=(r_1^{cur}, \dots, r_n^{cur})$.

In this local search algorithm, n different local neighborhoods $N_1(r^{cur}), \dots, N_n(r^{cur})$ are defined for each solution r^{cur} , and each local neighborhood $N_i(r^{cur})$ contains all the solutions that can be reached by executing one of the move operators once. In other words, $N_i(r^{cur})$ includes all the solutions obtained by enumerating all changes in one component r_i of the current solution r^{cur} . Note that the current solution r^{cur} is also included in its own neighborhood.

Navigation strategy describes the selection of the next iterative solution from the local neighborhood. Several navigation strategies can be chosen, such as best improvement, first improvement [110] and so on. A better iterative solution can speed up convergence of the local search algorithm to an optimal solution, but it often requires a higher computational cost for evaluation and comparison. Given the definition of the local neighborhood $N_i(r^{cur})$ and the cost function $F_p(r; \mu)$, the best improvement navigation strategy can be employed in the local search algorithm provided that $\forall i, j \in \{1, \dots, n\}$ and $j \neq i$, $\exists x_i \in \mathbf{R}^+$ and $y_{i,j} \in \mathbf{Z}^+$ such that $F_p(r; \mu)$ for $r \in N_i(r^{cur})$ is monotonically non-decreasing with the function values of the partial function $F_{q,i}(r; \mu, x, y)$ given by

$$F_{q,i}(r; \mu, x, y) = x_i * f_i(r) + \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r)\} \quad (27)$$

Where x and y are two general parameters and $f_i(r)$ for $i \in \{1, \dots, n\}$ is a partial objective function that measures the total cost of the unit actions in the path r_i . This is because an optimal solution can be found from $N_i(r^{cur})$ with respect to the function in (27) by solving a single agent path planning problem for the agent A_i with computationally efficient shortest path algorithms like the Dijkstra or A* algorithm. In this single agent path planning problem

for the agent A_i , there exist many equivalent paths that only differs in T_i for each component r_i , which translates to many equivalent solutions in neighborhood $N_i(r^{cur})$. A simple way to eliminate the redundant equivalent solutions in $N_i(r^{cur})$ is to set a bound on T_i . Since it is unknown what lowest value of T_i could ensure the existence of feasible solutions, setting an upper bound on T_i may prevent a feasible solution from being found. By contrast, setting a lower bound lb_i on T_i does not give rise to this problem since there always exist an equivalent path r'_i for r_i with T'_i greater than lb_i , the only downside is the high evaluation cost due to a high lower bound. In this case, the lower bound lb_i on T_i is dynamically set to the last time step when other agents occupy the destination of the agent A_i before $T_{-i} = \max_{1 \leq j \leq n, j \neq i} (T_j)$. Then the single agent path planning problem is to find a path r_i with the lowest $F_{q,i}(r; \mu, x, y)$ in (27), which has the total time step T_i greater than lb_i . This can be easily done using the A* algorithm with a time-dependent heuristic function H . Before the time step lb_i , $H(n) = x_i * (lb_i - t(n))$, where $t(n)$ returns the time step of the node n . After lb_i , $H(n) = x_i * \text{Manhattan}(n, n_g)$, where the function $\text{Manhattan}(n, n_g)$ returns the Manhattan distance from the node n to the goal node n_g . Tie breaking can be done by selecting the node with the lowest $t(n)$.

In summary, beginning with an initial solution $r^{[0]}$, the local search algorithm obtains each subsequent solution $r^{[\tau]}$ ($\tau=1,2,\dots$) by selecting an improving solution from $N_i(r^{[\tau-1]})$ in a systematic manner with respect to the cost function $F_p(r; \mu)$ in (25) until a locally optimal solution is reached. This iterative algorithm is called Iterative Path Coordination (IPC). The pseudo-code of IPC is given in Figure 3-1. Note that r^{best} is the global variable that records the best MAPP solution found so far and it is initially set to empty. The function $\text{FindImprovingSolution}(N_i(r^{[\tau-1]}), F_p(r; \mu))$ returns an improving solution with respect to

the cost function $F_p(r; \mu)$ in the neighborhood $N_i(r^{[\tau-1]})$, which is obtained by applying changes in the component r_i of the current solution $r^{[\tau-1]}$. The best improvement navigation strategy can be employed in **FindImprovingSolution** to find an optimal MAPP solution from $N_i(r^{[\tau-1]})$ provided that $\forall i, j \in \{1, \dots, n\}$ and $j \neq i$, $\exists x_i \in \mathbf{R}^+$ and $y_{i,j} \in \mathbf{Z}^+$ such that $F_p(r; \mu)$ for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function value of the function $F_{q,i}(r; \mu, x, y)$ in (27). The function **pcUpdate**() sets the value for the penalty coefficient μ using the parameter setting strategies discussed in the next section 3.2.2.

Algorithm 3-1: IPC (r^{init}, r^{best}, F_p)

- 1) $r^{[0]} \leftarrow r^{init}$;
 - 2) $i \leftarrow 1$ and $\tau \leftarrow 1$;
 - 3) Repeat the following procedure until no further improvement is possible:
 - a) $\mu \leftarrow \text{pcUpdate}()$;
 - b) $r^{[\tau]} \leftarrow \text{FindImprovingSolution}(N_i(r^{[\tau-1]}), F_p(r; \mu))$;
 - c) $\tau \leftarrow \tau + 1$;
 - d) if $i \geq n$, then $i \leftarrow 1$, or else $i \leftarrow i + 1$;
 - e) if $r^{[\tau-1]}$ is feasible and $F(r^{best}) > F(r^{[\tau-1]})$, or $r^{[\tau-1]}$ is feasible and $r^{best} = \emptyset$, then $r^{best} \leftarrow r^{[\tau-1]}$;
 - 4) Return $r^{[\tau-1]}$;
-

Figure 3-1. Pseudocode for the Iterative Path Coordination (IPC) Algorithm for MAPP.

3.2.2 Parameter Setting Strategies for Penalty Coefficient μ

Three parameter setting strategies are provided in this section, namely the fixed μ strategy, incremental μ strategy and adaptive μ strategy.

Fixed μ strategy - The penalty coefficient μ scales the penalty cost in the cost function $F_p(r; \mu)$ in (25), so a relatively high value of μ increases the relative influence of the penalty

term and therefore results in a biased search towards feasible solutions. As a result, when μ is set to a high enough value such that the penalty term dominates the cost function $F_p(r; \mu)$, the local search algorithm will only focus on reducing the number of collisions within the current solution and therefore the violations of dynamic constraints in (2b) can be suppressed. Thus, the fixed μ strategy needs to set μ to a sufficiently high fixed value in order to bias towards the primary goal of finding feasible solutions.

Unfortunately, setting μ to a high enough value reduces the considerations for higher quality solutions. As the solution quality is also of concern in this work, another parameter setting strategy called incremental μ strategy is proposed.

Incremental μ strategy - The incremental μ strategy initially sets μ to a small value ε_0 and then increases its value with each iteration. In doing so, the concern with solution quality will be addressed at the beginning and then gradually weakened with further iterations. In this work, a scaling scheme $\mu_{\tau+1} \leftarrow \sqrt[n]{a_l} * \mu_\tau$ is adopted for incremental μ strategy, where n is the number of agents, a_l is a scaling rate and τ is the iteration number in the local search algorithm for MAPP.

Though the incremental μ strategy may be better than fixed μ strategy in guiding IPC to find a superior MAPP solution, it is not adaptive to different MAPP scenarios as it adopts a monotonically increasing update of μ . To address this problem, another parameter setting strategy for μ called adaptive μ strategy is proposed.

Adaptive μ strategy - The adaptive μ strategy dynamically sets the value of μ according to the current solution or the historical information obtained during the search process. By taking into account the current solution, the value of μ can be updated according the function given by

$$\mu = F(r) / \{\lambda * \Sigma(r)\} \quad (28)$$

where $\Sigma(r)$ measures the penalty cost within the solution r and for the cost function $F_p(r; \mu)$ in (25) it is given by

$$\Sigma(r) = \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{\Gamma_{i,j}(r)\} \quad (29)$$

In the function (28), λ is a positive weight parameter that controls relative influence of the objective function $F(r)$ and the penalty cost $\Sigma(r)$. When $\lambda=0.5$, the influence of $F(r)$ and $\Sigma(r)$ is balanced.

Definition 13. *Locally optimal basin* – is the basin of attraction of a local optimum.

Definition 14. *Infeasible locally optimal basin* – is the local optimal basin around an infeasible local optimum.

The idea behind the adaptive μ strategy is to increase the value of μ when the penalty cost $\Sigma(r)$ is observed to have increased after some ω iterations. This has the effect of biasing the search of IPC towards feasible solution regions. However, when the penalty cost is observed to have decreased after the same ω iterations, the search is now entering a locally optimal basin. Under these circumstances, the value of μ should be decreased so that the search within the locally optimal basin would place greater emphasis on the objective function $F(r)$ and thereby increase the probability of finding a more superior MAPP solution. In order to realize this design, the value of λ in (28) must be modified appropriately. Given the current solution r^{cur} and the last solution r^{last} ω iterations before, the parameter λ can be updated according to the function given by

$$\lambda_{m+1} = \begin{cases} 0.5, & \text{if } r^{cur} \in R^f \\ \lambda_m * \{\Sigma(r^{last}) / \Sigma(r^{cur})\}^{b_\lambda}, & \text{otherwise} \end{cases} \quad (30)$$

where $\lambda \in [0.00001, 0.5]$, $m=0, 1, 2, \dots$, and b_λ is an exponential factor that scales the changes of the penalty cost between the current solution r^{cur} and the last solution r^{last} . Since the relative influence of the objective function $F(r)$ is higher than the penalty term $\Sigma(r)$ when $\lambda > 0.5$, so an upper bound $\lambda_{max}=0.5$ is imposed to guarantee the convergence of IPC towards a feasible solution. At the same time, a lower bound $\lambda_{min}=0.00001$ is imposed to prevent the value of μ in (28) from going to infinity when λ reaches 0.

However, when searching in the feasible region R^f , the penalty cost $\Sigma(r^f)$ of the feasible solution r^f is 0 and therefore μ will be set to infinity in (28). To address this problem, the scaling scheme $\mu_{m+1} \leftarrow a_A * \mu_m$ is used when $r \in R^f$ in the adaptive μ strategy, where a_A is a positive value less than 1. By gradually decreasing the value of μ when searching in the feasible region, the solution search can be made to navigate out of this attraction basin around a feasible locally optimal solution when further promising MAPP solutions can no longer be found. To avoid setting a too low value for μ , a lower bound μ_{min} is imposed and it is set to the mean value of the cost for a move unit action. In the adaptive μ strategy, at every ω iterations, the value of λ is updated according to function (30) while the value of μ is updated according to the function given by

$$\mu_{m+1} = \begin{cases} a_A * \mu_m, & \text{if } r^{cur} \in R^f \text{ and } \mu_m \geq \mu_{min} \\ F(r^{cur}) / \{\lambda_m * \Sigma(r^{cur})\}, & \text{if } r^{cur} \notin R^f \end{cases} \quad (31)$$

where $m=0,1,2,\dots$. The iteration counting parameter ω is set to the value of $5*n$, where n is the number of agents. The suitable values for scaling rate a_A in (31) and the exponential factor b_λ in (30) were empirically determined to be 0.1 and 2.0 respectively and were used in all the experiments presented.

3.3 Solving MAPP Optimization Problems using IPC

The following sections will describe how the total cost and makespan minimized MAPP problems introduced in section 1.3 can be solved by using the proposed IPC algorithm.

3.3.1 Iterative Path Coordination for Makespan Minimized MAPP

Since the objective function $F_m(r)$ of the makespan minimized MAPP problem in (3) is a minimax function, unconstrained optimization algorithms like IPC can not be directly applied to solve this non-smooth optimization problem. Previous line search approaches ([87], [88]) or trust region methods ([89], [90], [91]) try to approximate the optimal solution of the minimax optimization problem by solving its equivalent inequality constrained problem ([92], [93], [94]) or its smooth approximate problem ([102], [103], [104], [105]). A smooth approximation to the minmax function can be obtained by using the regularization techniques like maximal entropy function [95] [96]. Adopting the maximal entropy function for the objective function $F_m(r)$ in (3), then the approximate problem is given by

$$\min_r F_e(r; c) = (1/c) \ln \{ \sum_{i=1}^n \exp(c * \Phi_i(r)) \} \quad (32)$$

Subject to (2a) and (2b), where c is a control parameter, $\Phi_i(r)$ given in (3) is the problem specific function that calculates the total time related cost of all unit actions in the path r_i . As $c \rightarrow +\infty$, $F_e(r; c) \rightarrow F_m(r)$. As a result, the minimization problem in (32) is equivalent to the makespan minimized MAPP problem in (3) when c is set to a sufficiently high value. Like the minimax problem in (3), the minimization problem in (32) with a sufficiently high value of c may have neutral areas [99] or plateau in the fitness landscape since the partial objective function $\Phi_i(r)$ with the maximal value dominates the objective function in (32). In this case, the descend information is not available. To address this problem, an incremental scheme is recommended for the control parameter c , in which c is initially set to a small non-negative

value and then gradually increased at each iteration. In so doing, the solution search can be gradually and directionally steered towards an optimal makespan MAPP solution. As a result, the optimal solution of the makespan minimized MAPP problem can be approximated by solving a series of minimization problems (32) with gradually increasing c .

The minimization problem in (32) with a specific value of c can be solved by using the feasible direction method [51], which moves to an improving feasible solution from the current feasible solution along one of the descent feasible directions. As such, for any two iterative feasible solution $r^{(k-1)}$ and $r^{(k)}$ such that $F_e(r^{(k)}; c) < F_e(r^{(k-1)}; c)$, a local minimum of the minimization problem in (32) can be eventually obtained as the iteration k continues. To use the feasible direction method for the minimization problem in (32), two issues need to be addressed. One is the descent feasible direction d , which points out the possible improving feasible solutions. The other is the step length, which is related to the selection of the improving feasible solution along the search direction d .

The descent feasible direction d is generally required to satisfy constraints (2a), (2b) and the condition given by

$$\nabla F_e(r; c) * d < 0 \quad (33)$$

where $\nabla F_e(r; c)$ is the gradient of the objective function $F_e(r; c)$ over the variables $x_i = \Phi_i(r)$ for $i \in \{1, \dots, n\}$ and is given by

$$\nabla F_e(r; c) = \sum_{i=1}^n \{ [1 / \sum_{j=1}^n \exp(c * [\Phi_j(r) - \Phi_i(r)])] * \overrightarrow{\Phi_i} \} \quad (34)$$

where $\overrightarrow{\Phi_i}$ is the coordinate vector associated with the variable $x_i = \Phi_i(r)$ ($i \in \{1, \dots, n\}$). As a result, at the iteration k of the feasible direction approach for the minimization problem in

(32), given the current feasible solution $r^{(k-1)}$, a new solution r' is considered a possible improving feasible solution when it satisfies the constraints (2a), (2b) and the condition

$$\sum_{i=1}^n \{ \Phi_i(r') / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]) \} < \sum_{i=1}^n \{ \Phi_i(r^{(k-1)}) / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]) \} \quad (35)$$

The next issue is selecting an improving feasible solution as the iterative solution $r^{(k)}$. It is desirable to choose the solution that gives the maximal reduction to the objective function $F_e(r)$, but finding this optimal solution could be intractable. Alternatively, one could just find an improving feasible solution near the last iterative solution $r^{(k-1)}$. This is because the convergence of the feasible direction method is usually guaranteed under a sufficiently small step length. As a result, given an starting solution $r^{(1)}$, the iterative solution $r^{(k)}$ can be obtained by solving the following problem with a local search algorithm once the condition (35) is satisfied.

$$\min_r F_s(r; c, r^{(k-1)}) = \sum_{i=1}^n \{ \Phi_i(r) / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]) \} \quad (36)$$

Subject to (2a) and (2b). A problem in solving the minimization problem in (36) using the feasible direction method is that the starting solution $r^{(1)}$ and the iterative solutions $r^{(k)}$ are all required to be feasible. To address this problem, a good starting solution $r^{(1)}$ can be obtained by solving the minimization problem in (36) with the control parameter $c=0$ and a given initial solution $r^{(0)}$. Here the initial solution $r^{(0)}$ does not need to be feasible. In the implementation, $r^{(0)}$ is initialized to r^{init} , which can be generated randomly or heuristically. In this work, r^{init} is obtained by planning an individual path for each agent with all other agents assumed absent.

Figure 3-2. Overview description of the Iterative Makespan Minimization (IMM) algorithm for solving makespan minimized MAPP.

In summary, as shown in Figure 3-2, given an initial solution $r^{(0)}$, which is given by r^{init} , the optimal solution of the makespan minimized MAPP problem can be approximated by solving a sequence of minimization problem in (36) with the value of the control parameter c gradually increasing from 0 based on an increment rate of ε_c until no further improvement can be made within the iterative solution $r^{(k)}$ or a preset time bound is reached. At each iteration k , the minimization problem in (36) is solved by the algorithm $\mathbf{IPC}(r^{(k-1)}, r^{best}, F_p)$ given in Figure 3-1 with the problem specific cost function given by

$$F_p(r; \mu) = F_S(r; c, r^{(k-1)}) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \Gamma_{i,j}(r) \quad (37)$$

where $\Gamma_{i,j}(r)$ given in (26) measures the violations of the dynamic constraints (2b) between the paths r_i and r_j .

The function value of $F_p(r; \mu)$ in (37) for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function $F_{q,i}(r; \mu, x, y)$ in (27) when the partial objective function $f_i(r)$ and the general parameters x and y in (27) are set as follows

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Phi_i(r) \quad (38)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1 / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]), \quad (39)$$

$$y_{i,j} = 1$$

where c is the control parameter in (32). The problem-specific function $\Phi_i(r)$ given in (3) calculates the total time related cost of all unit actions in the path r_i . As a result, at each iteration τ of $\mathbf{IPC}(r^{(k-1)}, r^{best}, F_p)$, each iterative MAPP solution $r^{[\tau]}$ is obtained by employing the best improvement navigation strategy in the function **FindImprovingSolution**. The iterative algorithm using IPC for makespan minimized MAPP is called *Iterative Makespan Minimization* (IMM).

3.3.2 Iterative Path Coordination for Total Cost Minimized MAPP

The total energy cost minimized MAPP problem in (4) is a difficult minimization problem as many alternative solutions with equal objective values exist due to the assignment of zero traversal cost for the idle unit action. This results in neutral areas [99] or plateau in the fitness landscape. To address this problem, the idea of using a second optimality criterion to navigate the search when being trapped in the neutral area is explored. Since a preferred solution is one with as few idle actions as possible, the second objective function that can be employed is given by

$$F_2(r) = \sum_{i=1}^n I_i(r) \quad (40)$$

where the function $I_i(r)$ calculates the total costs of all idle unit actions in the path r_i . To deal with the MAPP optimization problem with two optimality criteria [106], a way is to aggregate the two objective functions into a weighted function [107], [113] given by

$$F_w(r; \rho_c) = F_c(r) + \rho_c * F_2(r) \quad (41)$$

where ρ_c is the cost ratio between the energy cost $F_c(r)$ in (4) and the idle action costs $F_2(r)$, and $\rho_c > 0$. Since the objective function $F_c(r)$ has a higher priority over $F_2(r)$, ρ_c is required to be below 1 such that $F_c(r)$ has a relatively higher influence than $F_2(r)$ in (41). The optimal solution of the total energy cost minimized MAPP problem can now be approximated by solving the problem given by

$$\min_r F_w(r; \rho_c) = F_c(r) + \rho_c * F_2(r) \quad (42)$$

Subject to (2a), (2b) and $0 < \rho_c < 1$.

Figure 3-3. Overview description of the Iterative Cumulation Minimization (ICM) algorithm for solving total cost minimized MAPP.

Thus, as shown in Figure 3-3, beginning with the initial solution $r^{(0)}$, which is given by r^{init} , the optimal solution of the total energy cost minimized MAPP problem is approximated by solving the minimization problem in (42) until no further improvement is possible within the current solution or a preset time bound is reached. At each iteration k , the minimization problem in (42) is solved by the algorithm **IPC**($r^{(k-1)}, r^{best}, F_p$) in Figure 3-1 with the problem specific cost function $F_p(r; \mu)$ given by

$$F_p(r; \mu) = F_w(r; \rho_c) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \Gamma_{i,j}(r) \quad (43)$$

where $\Gamma_{i,j}(r)$ given in (26) measures the violations of the dynamic constraints (2b) between the paths r_i and r_j .

The function value of $F_p(r; \mu)$ in (43) for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function $F_{q,i}(r; \mu, x, y)$ in (27) when the partial objective function $f_i(r)$ and the general parameters x and y in (27) are set as follows

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Psi_i(r) + \rho_c * I_i(r) \quad (44)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1, y_{i,j} = 1 \quad (45)$$

where ρ_c is the cost ratio in (41). The problem specific function $\Psi_i(r)$ given in (4) calculates the total distances traversed by the agent A_i . As a result, at each iteration τ of **IPC**($r^{(k-1)}, r^{best}, F_p$), $r^{[\tau]}$ is obtained by employing the best improvement navigation strategy in the function **FindImprovingSolution**. The iterative algorithm using IPC for total energy cost minimized MAPP is called Iterative Cumulation Minimization (ICM).

3.4 Experimental Evaluation

The proposed Iterative Path Coordination (IPC) algorithm for MAPP is evaluated in a simulation environment consisting of a 30×20 grid map as shown in Figure 3-4. Such grid maps were similarly employed during experimental evaluation in works such as [24], [26], [39] and [40]. Each unit agent is assigned different starting and target positions. The unit time interval is set to dt , which is the time taken by executing one moving or idle unit action.

Figure 3-4. A random scenario in a 30x20 grid map with 120 agents. Static obstacles occupying 20% of all accessible grid nodes are denoted by shadowed areas. The small grey squares denote the mobile agents and the dashed squares denote their respective unique destinations.

The simulation program was coded in the C++ programming language using the Visual Studio 2010 software development platform and executed on an Intel(R) core™ 2 quad (2.83 GHz) with 3.25 GB of working memory.

To investigate the performance and relative merit of the proposed local search algorithms in this thesis, four performance criteria were adopted.

- a) *Success rate* --- the percentage of time in the 100 random scenarios all the n agents found collision-free paths to their respective destinations. This is the most important performance measure as all other measures are irrelevant if the n agents are unable to complete their respective goals.

- b) *Task completion time* --- the average time taken to complete the movement of all n agents to their respective destinations in all successful random scenarios in a). This is the optimization objective set out in makespan minimized MAPP and will determine how fast the proposed algorithm is able to get the required task completed.
- c) *Cumulative cost* --- the average total traversal cost incurred by the movement of all n agents to their respective destinations in all successful random scenarios in a). This is the optimization objective set out in total cost minimized MAPP and will show how energy efficient the proposed algorithm is in getting the required task completed.
- d) *Computational time* --- the average time taken by the simulation code to compute the viable paths for all n agents in the successful random scenarios in a). This criterion provides a good empirical measure of the time complexity of the proposed algorithm.

3.4.1 Evaluation of IPC-based Algorithms with Increasing Allowable Computational Time

The penalty coefficient μ in the IPC algorithm scales the penalty term in the cost function (25), and may therefore influence IPC's convergence towards a feasible solution. As for the two IPC-based algorithms for solving makespan and total cost minimized MAPP, the control parameter c influences the performance of IMM in finding a superior makespan minimized MAPP solution, while the cost ratio ρ_c controls ICM in finding an energy efficient MAPP solution. The following sections evaluate these parameters in influencing the performance of IPC, IMM and ICM. Since the IPC-based algorithms are iterative algorithms that can improve the solution quality with increasing computational time, the performance of these algorithms were evaluated individually against the allowable computational time in a 30x20

grid map with an fixed agent count of 120 and static obstacle coverage of 20%, as shown in Figure 3-4.

3.4.1.1 Evaluation of IPC under different penalty coefficient μ setting strategies

The following experiments evaluate the different penalty coefficient μ setting strategies of the IPC algorithm when used within the IMM algorithm for solving makespan minimized MAPP. Three different μ setting strategies, namely fixed μ strategy, incremental μ strategy and adaptive μ strategy, were evaluated. For the fixed μ strategy, a low value of $\mu=0.1$ and a high value of $\mu=100$ were investigated. Two different scaling rates of $a_l=10$ (μ low-rate) and $a_l=1000000$ (μ high-rate) were used in incremental μ strategy. The initial small starting value of μ given by ε_0 was set to $1/n$. Finally, the adaptive μ strategy updates the value of μ according to the function (31) with the value settings of $a_A=0.1$, $\mu_{min}=1/n$ and $\omega=5*n$, $\lambda_0=0.05$, $\mu_0=\mu_{min}$. Other parameter ε_c for control parameter c in Figure 3-2 is set to 1. The number of agents n used in these experiments was 120.

Figure 3-5. Success rate of IMM with different μ setting strategies against allowable computational time.

Figure 3-6. Task completion time of IMM with different μ setting strategies against allowable computational time.

A high value of μ will increase the relative influence of the penalty term in the cost function $F_p(r; \mu)$ given in (25), which will effectively reduce the relative influence of the objective function $F(r)$ with respect to the importance of finding feasible solutions. As expected, a higher fixed value of μ results in a faster local convergence of IPC towards a feasible solution but reduces the probability of finding a superior solution. As a result, IMM produced better success rate but a poorer task completion time performance at the high value (Fixed $\mu=100$) compared to the low value (Fixed $\mu=0.1$) (see Figures 3-5 and 3-6). Since the incremental μ strategy gradually increases the value of μ to a high value with the same initial value ε_0 , a high μ increment rate leads quickly to a high value of μ and therefore results in similar performance characteristics to that of a high fixed value of μ . However, like the fixed μ strategy, both (μ high-rate) and (μ low-rate) suffers the success rate versus task completion time tradeoff.

In contrast, the adaptive μ strategy increases the value of μ when the penalty cost increases and decreases it otherwise. In so doing, the adaptive μ strategy can lead to a faster

convergence rate towards feasible solutions and also performs an intensive search within every attraction basin around locally optimal solutions for a superior MAPP solution. It was observed that the proposed adaptive μ strategy produced success rate performance that was as good as the high μ (Fixed $\mu=100$) while still able to achieve task completion time that were close to those of the low μ (Fixed $\mu=0.1$). Subsequent evaluations will use the adaptive μ strategy.

3.4.1.2 Evaluation of IMM under different increment rate ε_c for control parameter c

Since the additional control parameter c is used in IMM to navigate the search for a makespan minimized MAPP solution, the following experiments evaluate different increment rates ε_c (see Figure 3-2) for increasing the value of control parameter c in influencing the performance of IMM. Three different values of $\varepsilon_c=0.0001$, $\varepsilon_c=1$ and $\varepsilon_c=100$ with the parameter setting of adaptive μ strategy ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for IPC were used in the evaluation.

Figure 3-7. Success rate of IMM with different increment rates ε_c against allowable computational time.

Figure 3-8. Task completion time of IMM with different increment rates ϵ_c against allowable computational time.

As the starting feasible solution $r^{(1)}$ is obtained by solving the problem in (36) with the same value of $c=0$ and the same initial solution $r^{(0)}$, so the different values of increment rate ϵ_c had negligible influence on the success rate performance of IMM (see Figure 3-7). However, since a high value for the control parameter c could lead to weak descent information, employing a high increment rate of ϵ_c that quickly increases the value c will result in a slower convergence rate of IMM towards a locally optimal MAPP solution. As observed in Figure 3-8, given the same starting feasible solution $r^{(1)}$, a high value of ϵ_c resulted in a higher task completion time in the limited time bound of 10s. However, the approximate problem in (32) is equivalent to the makespan minimized MAPP problem when c is sufficiently high, so setting ϵ_c to a higher value leads to a more biased search towards a makespan minimized MAPP solution and therefore results in a better task completion performance of IMM if enough computational time is permitted (see Figure 3-8). It was observed that as the computational time goes beyond 60s, $\epsilon_c=100$ resulted in the better task completion time performance compared to the low value of $\epsilon_c=0.0001$. Compared with the

low $\varepsilon_c=0.0001$ and the high $\varepsilon_c=100$, the intermediate $\varepsilon_c=1$ leads to a neither too slow nor too fast increase of the control parameter c . Thus, it was observed that $\varepsilon_c=1$ converges most quickly to a superior task completion time performance as the computational time goes beyond 10s.

3.4.1.3 Evaluation of ICM under different cost ratios ρ_c

The following experiments evaluate the performance of ICM in solving the total cost minimized MAPP problem. Three fixed cost ratio values of $\rho_c=0.01$, $\rho_c=0.1$ and $\rho_c=1.0$ were used to investigate whether a low value of ρ_c yields a more energy efficient MAPP solution compare to a high value of ρ_c . The evaluations were done with the parameter setting of adaptive μ strategy ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for IPC.

Figure 3-9. Success rate of ICM with different cost ratios against allowable computational time.

Figure 3-10. Cumulative cost of ICM with different cost ratios against allowable computational time.

Figure 3-11. Task completion time of ICM with different cost ratios against allowable computational time.

Since the cost ratio ρ_c controls the relative influence of the energy costs incurred by move actions and the idle action costs, as ρ_c decreases, idle actions will be preferred over the move actions. Since the idle unit actions produce less collision constraints compared with the move unit action, a low value of ρ_c should result in a higher success rate than a high value if

enough computational time is allowed. This was indeed observed in Figure 3-9 where $\rho_c=0.1$ and $\rho_c=0.01$ had better success rate performances than $\rho_c=1.0$ for the allowable computational time of 200s. However, the lower cost ratio ρ_c increases the number of feasible solution evaluated during each iteration of IPC and is therefore expected to slow the increase of ICM's success rate with allowable computational time (see Figure 3-9). Given more computational time, the success rate of $\rho_c=0.01$ could increase further since its trajectory has not yet flatten.

When move action is preferred over idle action (ie. $\rho_c=1.0$), the agents will quickly approach their destinations along their desired shortest paths. To avoid the possible collisions along their paths, the tendency of choosing move action will make the agents detour from their shortest paths, thus leading to increasing energy consumption. In contrast, the tendency of choosing idle action allows the agent to stay still at some suitable places along its desired shortest path and therefore reduces the probability of detouring from the preferred shortest path. Since idle actions do not consume energy, unlike move actions, a low cost ratio ρ_c significantly reduces the cumulative cost (see Figure 3-10). It seems that the low cost ratio values like $\rho_c=0.1$ and $\rho_c=0.01$ are ideal choices for implementing total cost minimized MAPP. However, preference for idle actions does come with a side effect. If the MAPP task needs to be completed by the agents in the shortest possible time, the excessive use of idle actions will result in poorer task completion time performance as shown in the results in Figure 3-11. Unfortunately, the makespan minimized and total cost minimized MAPP optimization objectives do often conflict with each other.

3.5 Summary

This chapter proposed a basic local search algorithm called Iterative Path Coordination (IPC) for MAPP. The proposed IPC algorithm iteratively finds a lowest cost path for one of the agents with respect to a cost function given in (25) until no further improvement is possible within the current MAPP solution or a preset time bound is reached. To improve IPC's ability in finding a good MAPP solution, different parameter tuning strategies for the penalty coefficient μ are provided. To apply the proposed local search algorithm IPC for solving two challenging MAPP minimization problems, namely makespan minimized and total cost minimized MAPP, two different iterative algorithms called Iterative Makespan Minimization (IMM) and Iterative Cumulation Minimization (ICM) were then proposed. In the end, the performance of IPC was evaluated when being used within its individual iterative framework for solving the makespan minimized and total cost minimized MAPP problems under different parameter settings. It can be observed in Figures 3-5, 3-7 and 3-9 that it is difficult for IPC to achieve 100% success rate even if enough computational time is permitted. This is because IPC suffers from the problem of being trapped in a locally optimal basin, which may be infeasible in a crowded environment. In the following chapter, a variant of IPC that is able to solve the problem of being trapped in an infeasible locally optimal basin is proposed. Based on the observed results of evaluated parameters so far, the parameter settings of adaptive μ strategy ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$), $\varepsilon_c=1$ for control parameter c and the cost ratio $\rho_c=0.1$ will be used in the subsequent chapters

4 ITERATIVE PATH COORDINATION USING GUIDED SEARCH

In chapter 3, the weaknesses of IMM and ICM, the two IPC-based algorithms, in achieving good success rate performance in moderately crowded environments were observed. This is because the local search process has a tendency of getting trapped in an infeasible locally optimal basin. One way to address this problem is to use some guided techniques to navigate the search out of such a locally optimal basin. This chapter first reviews the use of guided techniques in local search methods and then proposes an improved variant of the IPC algorithm that incorporates guided local search. The proposed algorithm is then compared with the earlier proposed IPC-based algorithms.

4.1 Guided Search Techniques

The guided search techniques navigate the solution search by using historical information obtained during the search process. By using guided techniques in local search methods, the local search can be navigated out of the locally optimal basin. There are several local search algorithms that use guided techniques.

One such local search algorithm is tabu search [98]. It firstly builds a tabu list that memorizes the recently visited solutions, or the solution attributes. Then it forbids re-visiting these solutions in the tabu list or those solutions containing attributes such as violation of constraints. An issue associated with tabu list is that it may jeopardize the algorithm's ability of finding better solutions. For example, if one neighboring solution of a solution x is chosen to be visited next, then the current solution x will be added into the tabu list. In that case, the solution x is not allowed to be re-visited again. As a result, other unvisited neighboring

solutions of solution x may not be visited any more even if some of them can lead to better solutions. To address this problem, a strategy called aspiration criterion can be employed. In aspiration criterion, a tabulated solution can still be re-visited if it can lead to a better solution than the current best. The difficulty of applying tabu search to an optimization problem lies in the information recorded in the tabu list. A simple way of recording complete solutions in the tabu list could lead to large memory storage requirements, especially for large-scale MAPP problems.

Another local search algorithm using guided techniques is guided local search (GLS) [75], which uses a guided cost function in an attempt to help the local search escape from a locally optimal basin. GLS builds up this guided cost function by adding a solution feature penalty into the objective function. Each time a local optimum is reached, GLS modifies the cost function by increasing the penalty based on a specific scheme so as to help escape from the locally optimal basin. As such, the choice of the cost function and the way of modifying the cost function is important in navigating the solution search out of the locally optimal basin.

To use guided local search, solution features are first defined and each feature s_i for $i=1, \dots, m$ has a corresponding cost function g_i and a feature penalty h_i , which is set to 1 initially. In GLS, an augmented cost function is often used. Given the objective function $f(x)$ and m solution features, the augmented cost function is given by

$$F(x) = f(x) + \Lambda * \sum_{i=1}^m \{h_i(x) * D_i(x)\} \quad (46)$$

where Λ is a parameter and it is used to scale the penalty associated with the solution features, $D_i(x)$ is an indication of whether the solution feature s_i is present in the current solution x , $D_i(x)=1$ if s_i is present in the current solution x and 0 otherwise.

Each time a locally optimal solution x is reached, a utility function is calculated as follows

$$U(x, i) = D_i(x) * g_i(x) / h_i(x) \quad (47)$$

Then the feature penalty h_i of the feature s_i with the highest utility is increased. The increasing penalty of the features present in the local optimal solution will make the local optimal solution more costly than the surrounding solutions and will therefore help the algorithm escape from the locally optimal basin.

4.2 Guided Iterative Path Coordination (GIPC)

To help IPC escape from an infeasible locally optimal basin, this work adopts the guided local search technique and defines n solution features that are corresponding to the number of agents. Each solution feature s_i is defined as the violation of the dynamic constraints imposed by the path r_i . Then the solution feature penalty $h_i(r)$ associated with s_i is initially set to 1. Due to the sparse number of defined solution features for MAPP, the augmented cost function in (46) is not effective in distinguishing the solutions around the local optimum and may therefore make no difference in guiding the search out of the locally optimal basin. As a result, a guided cost function for MAPP is introduced here.

Guided cost function - The solution feature penalties are incorporated into the penalty term in the general cost function $F_p(r; \mu)$ in (25) to produce a general guided cost function for MAPP given by

$$F_g(r; \mu, h) = F(r) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{h_i(r) * h_j(r) * \Gamma_{i,j}(r)\} \quad (48)$$

Where μ is a non-negative penalty coefficient, $F(r)$ is a general objective function in a MAPP problem, $\Gamma_{i,j}(r)$ given in (26) measures the violations of the dynamic constraints in (2b) between the paths r_i and r_j . $h_i(r)$ is the solution feature penalty associated with the

solution feature s_i . Given the guided cost function in (48), the cost incurred by each solution feature s_i present in the current solution r is given by

$$D_i(r) * g_i(r) = 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{h_i(r) * h_j(r) * \Gamma_{i,j}(r)\} \quad (49)$$

When the infeasible locally optimal solution r^* is encountered, the utility function for each solution feature s_i is computed as

$$U(r^*, i) = \sum_{j=1, j \neq i}^n \{h_j(r^*) * \Gamma_{i,j}(r^*)\} \quad (50)$$

where $i = 1, 2, \dots, n$. The intuitive idea here is to make other agents give way to the agent with the most collisions in order to make further progress. μ and 0.5 is omitted in (50) since they are only used as scalars and make no difference in the comparisons of the utilities for all solution features. Thus, a guided mechanism that is used to modify the guided cost function is described as follows:

Guided mechanism - Each time a locally optimal solution is encountered and it is an infeasible solution, the utilities of all solution features are calculated using (50). Then the feature penalty of the solution feature with the highest utility is increased by 1.

Incorporating the guided cost function and guided mechanism into the IPC algorithm presented in the previous chapter, a new local search algorithm named *Guided Iterative Path Coordination* (GIPC) is proposed. This algorithm can terminate when a feasible locally optimal solution is reached or a preset time bound is reached. A locally optimal solution is deemed to be reached when the current solution is continuously kept for at least n iterations, where n is the number of agents. The pseudo code of GIPC is given in Figure 4-1. The function **FindImprovingSolution** ($N_i(r^{[\tau-1]})$, $F_g(r; \mu, h)$) returns an improving solution from the neighborhood $N_i(r^{[\tau-1]})$ with respect to the guided cost function $F_g(r; \mu, h)$ in (48). The

best improvement navigation strategy can be employed in **FindImprovingSolution** to find an optimal MAPP solution from $N_i(r^{[\tau-1]})$ provided that $\forall i, j \in \{1, \dots, n\}$ and $j \neq i$, $\exists x_i \in \mathbf{R}^+$ and $y_{i,j} \in \mathbf{Z}^+$ such that $F_g(r; \mu, h)$ for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function value of the function $F_{q,i}(r; \mu, x, y)$ in (27). The function $\maxIndex_{1 \leq j \leq n} \{U(r^*, j)\}$ returns the subscript index of the solution feature s_j with the maximal utility $U(r^*, j)$. The **pcUpdate**() scheme updates the value of μ according the adaptive μ strategy described in section 3.2.2. However, in the case of guided cost function $F_g(r; \mu, h)$, the penalty cost is given by

$$\Sigma(r) = \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{h_i(r) * h_j(r) * \Gamma_{i,j}(r)\} \quad (51)$$

Algorithm 4-1: GIPC ($r^{init}, r^{best}, F_g, U$)

- 1) $r^{[0]} \leftarrow r^{init}$, $h_j \leftarrow 1$ for $j=1, 2, \dots, n$;
 - 2) $i \leftarrow 1$, $\alpha \leftarrow 0$ and $\tau \leftarrow 1$;
 - 3) Repeat the following procedure until a feasible locally optimal solution is reached:
 - a) $\mu \leftarrow \text{pcUpdate}()$;
 - b) $r^{[\tau]} \leftarrow \text{FindImprovingSolution}(N_i(r^{[\tau-1]}), F_g(r; \mu, h))$;
 - c) If $r^{[\tau]} = r^{[\tau-1]}$, then $\alpha \leftarrow \alpha + 1$, or else $\alpha \leftarrow 0$;
 - d) If $\alpha \geq n$, then
 - i. $j_{max} \leftarrow \maxIndex_{1 \leq j \leq n} \{U(r^*, j)\}$;
 - ii. $h_{j_{max}} \leftarrow h_{j_{max}} + 1$.
 - iii. $\alpha \leftarrow 0$;
 - e) $\tau \leftarrow \tau + 1$;
 - f) if $i \geq n$, then $i \leftarrow 1$, or else $i \leftarrow i + 1$;
 - g) if $r^{[\tau-1]}$ is feasible and $F(r^{best}) > F(r^{[\tau-1]})$, or $r^{[\tau-1]}$ is feasible and $r^{best} = \emptyset$, then $r^{best} \leftarrow r^{[\tau-1]}$;
 - 4) Return $r^{[\tau-1]}$;
-

Figure 4-1. Pseudocode for the Guided Iterative Path Coordination (GIPC) algorithm.

4.3 Solving MAPP Optimization Problems using GIPC

4.3.1 Guided Iterative Makespan Minimization for Makespan Minimized MAPP

Figure 4-2. Overview description of the Guided Iterative Makespan Minimization (GIMM) algorithm for solving makespan minimized MAPP.

The iterative algorithm for solving the makespan minimized MAPP problem using GIPC is shown in Figure 4-2. Beginning with an initial solution $r^{(0)}$, which is given by r^{init} , the iterative algorithm approximates the optimal solution of the makespan minimized MAPP problem by solving a sequence of minimization problem in (36) with the value of the control parameter c gradually increasing from 0 based on an increment rate of ε_c until no further improvement can be made within the iterative solution $r^{(k)}$ or a preset time bound is reached. At each iteration k , the minimization problem in (36) is solved by the algorithm $\text{GIPC}(r^{(k-1)}, r^{best}, F_g, U)$ in Figure 4-1 with the problem specific guided cost function given by

$$F_g(r; \mu, h) = F_s(r; c, r^{(k-1)}) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{h_i(r) * h_j(r) * \Gamma_{i,j}(r)\} \quad (52)$$

where $F_s(r; c, r^{(k-1)})$ is the problem specific objective function in (36). The function $\Gamma_{i,j}(r)$ given in (26) measures the violations of the dynamic constraints (2b) between the paths r_i and r_j . $h_i(r)$ is the solution feature penalty associated with the solution feature s_i .

The function value of $F_g(r; \mu, h)$ in (52) for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function $F_{q,i}(r; \mu, x, y)$ in (27) when the settings of partial objective function $f_i(r)$ and the general parameters x and y in (27) are given by:

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Phi_i(r) \quad (53)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1 / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]), \quad (54)$$

$$y_{i,j} = h_i(r) * h_j(r)$$

where c is the control parameter in (32). The problem specific function $\Phi_i(r)$ given in (3) calculates the total time related cost of all unit actions in the path r_i . In this case, at each iteration τ of **GIPC**($r^{(k-1)}, r^{best}, F_g, U$) in Figure 4-1, each iterative MAPP solution $r^{[\tau]}$ is obtained by employing the best improvement navigation strategy in the function **FindImprovingSolution**. The iterative algorithm using GIPC for makespan minimized MAPP is called Guided Iterative Makespan Minimization (GIMM).

4.3.2 Guided Iterative Cumulation Minimization for Total Cost Minimized MAPP

Figure 4-3. Overview description of the Guided Iterative Cumulation Minimization (GICM) algorithm for solving total cost minimized MAPP.

The iterative algorithm for solving the total cost minimized MAPP problem using GIPC is shown in Figure 4-3. Beginning with the initial solution $r^{(0)}$, which is given by r^{init} , the optimal solution of the total energy cost minimized MAPP problem is approximated by iteratively solving the minimization problem in (42) until no further improvement is possible within the iterative solution $r^{(k)}$ or a preset time bound is reached. At each iteration k , the minimization problem in (42) is solved by the algorithm $\mathbf{GIPC}(r^{(k-1)}, r^{best}, F_g, U)$ in Figure 4-1 with the problem specific guided cost function $F_g(r; \mu, h)$ given by

$$F_g(r; \mu, h) = F_w(r; \rho_c) + 0.5 * \mu * \sum_{i=1}^n \sum_{j=1, j \neq i}^n \{h_i(r) * h_j(r) * \Gamma_{i,j}(r)\} \quad (55)$$

Unlike the guided cost function in (52), in the total cost minimized MAPP case, the objective function $F(r)$ is the problem specific objective function $F_w(r; \rho_c)$ given in (42).

The function value of $F_g(r; \mu, h)$ in (55) for $r \in N_i(r^{[\tau-1]})$ is monotonically non-decreasing with the function $F_{q,i}(r; \mu, x, y)$ in (27) when the partial objective function $f_i(r)$ and the general parameters x and y in (27) are set as:

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Psi_i(r) + \rho_c * I_i(r) \quad (56)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1, y_{i,j} = h_i(r) * h_j(r) \quad (57)$$

where ρ_c is the cost ratio in (41). The problem specific function $\Psi_i(r)$ given in (4) calculates the total distances traversed by the agent A_i . The problem specific function $I_i(r)$ given in (40) calculates the total cost of all unit actions in the path r_i . As a result, at each iteration τ of **GIPC**($r^{(k-1)}, r^{best}, F_g, U$) in Figure 4-1, each iterative MAPP solution $r^{[\tau]}$ is obtained by employing the best improvement navigation strategy in the function *FindImprovingSolution*. The iterative algorithm using GIPC for total cost minimized MAPP is called Guided Iterative Cumulation Minimization (GICM).

4.4 Experimental Evaluation

4.4.1 Comparative Study of GIPC-based Algorithms under Varying Agent Count

The following experiments compare the performance of GIPC-based algorithms with IPC-based algorithms under varying agent count in the 30x20 grid map shown in Figure 3-4. The varying agent count in the 30x20 grid map is obtained by changing the agent count from 0 to 200 with a fixed 20% static obstacle coverage. For each run with a different agent count, the same 100 random instances that have feasible solutions were used.

The comparisons between these MAPP algorithms were done under the same time bound of 5 minutes with the common parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and

$\mu_0=\mu_{min}$) for IPC and GIPC, $\varepsilon_c=1$ for control parameter c in IMM and GIMM, $\rho_c=0.1$ for ICM and GICM.

4.4.1.1 Comparative evaluation of GIMM for makespan minimized MAPP

The following experiments compare the performance of GIMM with IMM in solving makespan minimized MAPP on a 30x20 grid map.

Figure 4-4. Comparative success rates of IMM and GIMM under varying agent count in a 30x20 grid map.

Figure 4-5. Comparative task completion times of IMM and GIMM under varying agent count in a 30x20 grid map.

Figure 4-6. Comparative computational times of IMM and GIMM under varying agent count in a 30x20 grid map.

The increasing agent count quickly increases the probability of dynamic collisions between agents, which effectively translates to a lower possibility of feasible solutions in the solution space. As such, more iterations are required for iterative algorithms like IMM and GIMM to find a feasible solution, thus resulting in computational times that increase quickly with the increasing agent count (see Figure 4-6). The lower possibility of feasible solutions in the solution space also implies a higher possibility of IPC and GIPC getting into infeasible locally optimal basins. Since IPC can get trapped in an infeasible locally optimal basin, the corresponding success rate of IMM will drop quickly as the environment becomes moderately crowded (e.g. agent count > 120 in the 30x20 grid map), as was observed in Figure 4-4. On the other hand, since GIPC is designed with an additional guided technique, it is able to escape from an infeasible locally optima basin. This is the reason why GIMM was observed to achieve a better success rate performance than IMM (see Figure 4-4).

In order to study the relative performance in task completion time, computational time and cumulative cost (presented in the next section), only common successful instances from the

IPC-based and GIPC-based algorithms were employed. This ensures a fair comparison can be made between these two different approaches. Based on this method of comparison, it was observed that both the task completion time and computational time performances of GIMM and IMM were relatively similar (see Figures 4-5 and 4-6). This is because the guided technique employed in GIPC only helps the search process to escape from infeasible locally optimal basins. Its advantage in situation when a feasible solution has been found will not be realized. However, GIPC's strength can begin to be appreciated as the number of agents in the environment starts increasing and the search for feasible solutions becomes more challenging.

4.4.1.2 Comparative evaluation of GICM for total cost minimized MAPP

The next set of experiments investigates if GIPC can also improve GICM's ability to solve the total cost minimized MAPP problem compared to ICM. The same set of test scenarios for the 30x20 grid map was used.

Figure 4-7. Comparative success rates of ICM and GICM under varying agent count in a 30x20 grid map.

Figure 4-8. Comparative cumulative costs of ICM and GICM under varying agent count in a 30x20 grid map.

Figure 4-9. Comparative computational times of ICM and GICM under varying agent count in a 30x20 grid map.

Not surprisingly, similar improvement in the success rate performances was observed when GIPC was used to implement GICM (see Figure 4-7). Like before, the ability of the guided local search technique to navigate the search out of an infeasible locally optimal basin is particularly useful in crowded environments when searching for a cumulative cost minimized

solution. Like the results observed earlier in the previous section, GIPC does not result in significant improvement or degradation to both the cumulative cost and computational time performances of GICM relative to ICM (see Figures 4-8 and 4-9).

In summary, when compared with its equivalent IPC-based algorithms, GIPC exhibits superior success rate performance in finding MAPP solutions when used within GIMM and GICM for solving the makespan minimized and total cost minimized MAPP problems respectively. Though GIPC resulted in an improved success rate performance due to its guided local search technique, Figures 4-4 and 4-7 show that both GIMM and GICM were not able to continue maintaining a 100 % success rate as the agent count increased beyond 180 agents in the 30x20 grid map. The drop in success rate was due to the premature termination of these algorithms when the preset time bound of 5 minutes was reached (see time bound limit in Figures 4-6 and 4-9). Though the success rate performances of GIMM and GICM can be further improved if more computational time is permitted (see Figure 4-11), their computational times was observed to increase fairly rapidly with increasing agent count (Note: computational times are plotted in log scale in Figures 4-6 and 4-9). The computational time of the designed MAPP algorithm is a serious consideration in this work, as was outlined in the objectives discussed in section 1.2. The next section will further investigate how the different environment characteristics of the grid map influence the computational time performance of GIPC-based algorithms.

4.4.2 Evaluation of GIPC-based Algorithms' Computational Efficiency under Different Environment Characteristics

The computational time performance of GIMM and GICM is mainly dependent on two aspects of the GIPC algorithm. The first is the total number of iterations required by GIPC to

find a qualified MAPP solution. The second is the computational cost in searching for an improving solution within each iteration of GIPC. Both two measures are closely related to the agent count and the size of traversable space as they determine the size of solution space and the possibility of feasible solutions within it.

In section 4.4.1, the performance of GIPC-based algorithms was evaluated under varying agent count in the fixed-sized traversable space. In this section, the experiments will evaluate the performance of GIPC in finding a feasible MAPP solution under varying size of traversable space. For this purpose, only the GIMM algorithm for makespan minimized MAPP will be considered. Two methods were used to change the size of traversable space. The first varied the static obstacle density in a fixed-sized grid map. The second changed the size of the grid map. The evaluations were done by changing these two environment characteristics with the parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for GIPC and $\varepsilon_c=1$ for control parameter c in GIMM.

4.4.2.1 Evaluation of GIMM with different static obstacle density

The following experiments evaluate the success rate performance of GIMM against allowable computational time in the 30x20 grid map with a fixed agent count of 80 and the varying static obstacle density ranging from 0% to 40%.

Figure 4-10. Comparative success rates of GIMM with different static obstacle density in a 30x20 grid map.

Given the same fixed agent count of 80, the increasing static obstacles density reduces the size of traversable space and therefore leads to an increasing number of iterations required by GIPC to find a feasible solution. Like the increasing agent count scenario observed in section 4.4.1, increasing static obstacles density also resulted in increasing computational time. In other words, for a given allowable computational time, higher static obstacle density leads to poorer success rate (see Figure 4-10). Increasing map density by either increasing agent count or increasing static obstacle density will both lead to increasing computational time in the GIPC-based algorithms.

But which of these has a more significant influence on the computational time increase? The next set of experiments compares the relative influence of the agent count and static obstacle density on the computational time performance of GIMM in finding a feasible MAPP solution. The comparisons were done with different ratios of the agent count to the static obstacle density under the same map density of 53.33%. Given the map density of 53.33%, three different ratios, namely a low ratio of $\rho_d=80/40\%$, an intermediate ratio of $\rho_d=200/20\%$ and a high ratio of $\rho_d=320/0\%$, were used.

Figure 4-11. Comparative success rates of GIMM with different ratios of the agent count to static obstacle density in a 30x20 grid map.

It was observed in Figure 4-11 that it is more computationally costly for GIMM to find a feasible MAPP solution in an environment with a higher ratio of agent count to static obstacle density. Unlike the increasing agent count, the increasing static obstacle density does not increase the size of solution space. As a result, for a given map density, increasing static obstacle density does not increase the number of iterations required by GIPC to find a feasible MAPP solution as significantly as when the agent count is commensurately increased. Moreover, a higher static obstacle density reduces the size of the local neighborhood to be evaluated and therefore reduces computational cost in finding an improving neighboring MAPP solution at each iteration of GIPC. As a result of both these factors, increasing agent count has a more significant impact in increasing the computational cost of GIMM than increasing static obstacle density.

4.4.2.2 Evaluation of GIMM with different map size

The following set of experiments evaluates the computational time performance of GIMM in finding a feasible solution under different size of traversable space by varying the map size.

CHAPTER 4. ITERATIVE PATH COORDINATION USING GUIDED SEARCH

Three different sized grid maps were used, namely the small-sized 20x15, the intermediate-sized 30x20 and the large-sized 40x30 grid maps. For a fair and simple comparison, the same agent count of 160 with no static obstacle was used in all these three different map environments.

Figure 4-12. Comparative success rate of GIMM under varying map sizes against increasing iterations within GIPC.

Figure 4-13. Comparative computational time of GIMM under varying map sizes against increasing iterations within GIPC.

Figure 4-14. Comparative success rate of GIMM under varying map sizes against allowable computational time.

Given the same agent count of 160, the small-sized grid map has a lower size of traversable space and therefore leads to a higher map density and a higher probability of dynamic collisions between agents. Under such circumstances, more iterations within GIPC are required for GIMM to find a feasible MAPP solution in a smaller sized grid map (see Figure 4-12). However, the larger size of traversable space incurs a higher computational cost in searching for an improving neighboring MAPP solution within each iteration of GIPC, in this case, the computational time of GIMM in a larger sized grid map increases more quickly with increasing iterations of GIPC (see Figure 4-13). In short, the computational time performance of GIMM in finding a feasible solution is dependent on the total number of iterations required by GIPC to find a feasible solution and the computational cost in searching for an improving neighboring solution with each of these iterations. As a result, the computational time of GIMM increases as the map density and map size increases, and decreases as the map density and map size decreases. However, the overall computational time performance of GIMM is not changing monotonically with map size as map size and map density has an inverse relationship in the experiments presented. As can be observed in

Figure 4-14, compared with intermediate-sized map (30x20), the small-sized map (20x15) leads to a more significantly increase of the total number of iterations required by GIPC to find a feasible solution while the large-sized map (40x30) results in a more rapidly increasing computational cost with each iteration of GIPC. As a result, the success rate of GIMM in the intermediate-sized 30x20 grid map increases faster against allowable computational time than either the small (20x15) or the large (40x30) grid maps.

4.5 Summary

This chapter proposed a guided local search algorithm called Guided Iterative Path Coordination (GIPC) for MAPP, which is able to address IPC's problem of being trapped in infeasible locally optimal basin by modifying the guided cost function given in (48) according to a guided mechanism described in section 4.2. Compared with the iterative frameworks using IPC as shown in Figures 3-2 and 3-3, IPC is replaced with GIPC within the individual iterative framework for solving makespan minimized and total cost minimized MAPP problems in Figures 4-2 and 4-3. As such, two GIPC-based iterative algorithms namely Guided Iterative Makespan Minimization (GIMM) and Guided Iterative Cumulation Minimization (GICM) were proposed. The GIPC-based iterative algorithms were finally evaluated under different environment characteristics. It was observed that GIPC is superior to IPC in finding a feasible MAPP solution but it suffers from a quickly increasing computational time when the agent count and the size of traversable space are increasing. In a crowded environment with a large number of agents, the computational time of GIPC can be significantly high.

CHAPTER 4. ITERATIVE PATH COORDINATION USING GUIDED SEARCH

In the next chapter, a stochastic variant of IPC is proposed. By generating neighboring solutions probabilistically, the stochastic variant of IPC is able to both address the IPC's problem of being trapped in locally optimal basins and the computational time issue of GIPC.

5 ITERATIVE PATH COORDINATION USING STOCHASTIC SEARCH

Though GIPC has demonstrated a superior performance in escaping from infeasible locally optima basins, it is not computationally efficient in finding feasible MAPP solutions when the agent count and the size of traversable space increases. This chapter proposes a stochastic local search algorithm that probabilistically generates neighboring MAPP solutions. By generating solutions in a probabilistic way, the stochastic local search algorithm avoids performing an exact search in the local neighborhood and therefore reduces the computational cost in finding an improving neighboring MAPP solution compared with GIPC. Moreover, by allowing non-improving solutions to be accepted with a certain probability, the stochastic local search algorithm is also able to address the problem of being trapped in locally optimal basins. The following sections will firstly review the use of stochastic techniques for local search methods, and then introduce the proposed stochastic local search algorithm for MAPP. Finally, the proposed stochastic local search algorithm is evaluated and compared with the GIPC-based algorithms.

5.1 Stochastic Search Techniques

Stochastic search algorithms make random choices in generating or selecting candidate solutions at each search step. By using a random mechanism in the local search methods, there is a chance that the local search can escape from locally optimal basins. These local search algorithms using stochastic search are called stochastic local search algorithms.

Iterated local search (ILS) [76] is a stochastic local search algorithm that adopts a restart strategy for addressing the problem of being trapped in locally optimal basins. The restart

strategy perturbs the current solution each time a local optimum is encountered. As a result, the ILS algorithm iteratively applies the local search heuristic to find a local optimum from a starting point and then perturbs the current local optimum in order to obtain a new starting point. To escape from locally optimal basins, the perturbation strength should be sufficient so as to obtain a new starting point that can lead to a different locally optimal basin. A simple way of perturbing the solution is using a random sampling. However, this strategy will result in a poor performance of ILS as the problem instance size increases. This is due to the assumption that the local optima may be clustered together when the problem instance size is large. As a result, a biased sampling is desired for ILS when solving an optimization problem such as MAPP, which has a large problem instance size. In this case, designing an efficient biased sampling strategy is required for ILS.

An alternative and easier way of addressing the problem of being trapped in locally optimal basins in stochastic local search algorithms is using a probabilistic mechanism or scheme. This probabilistic mechanism generates neighboring solutions probabilistically. In this case, the non-improving solutions are allowed to be accepted with a certain probability and therefore the stochastic local search has a chance to escape from locally optimal basins. Simulated annealing [77] is one such stochastic local search algorithm for approximating the globally optimal solution of an optimization problem. It starts from an initial solution and moves to a neighboring solution by making a small change to the current solution. For example, by randomly exchanging the positions of any two sub-solutions in the current solution. It adopts a greedy strategy to store the best solution that is found. However, the greedy strategy may get trapped in locally optimal basins; so simulated annealing allows the selection of a worse neighboring solution with a small probability in order to escape from locally optimal basins. This algorithm terminates when there is no further update of the

current best solution, which means a locally or globally optimal solution has been found, or the temperature T is below a preset value. The stochastic local search algorithm for MAPP proposed in the next section adopts the probabilistic mechanism.

5.2 Probabilistic Iterative Path Coordination (PIPC)

To address the problem of being trapped in locally optimal basins in IPC, a probabilistic mechanism that probabilistically generates neighboring solutions is incorporated. To encourage the generation of the optimal solution, the better solution in the local neighborhood should ideally be assigned a higher generation probability. That is to say, for a MAPP minimization problem, the theoretical generation probability of each solution r in $N_i(r^{cur})$ would be monotonically non-increasing with the increasing value of the cost function $F_p(r; \mu)$ in (25). As a result, given the neighborhood $N_i(r^{cur})$ and the cost function $F_p(r; \mu)$, the probabilistic solution generation mechanism generates a neighboring solution from the local neighborhood $N_i(r^{cur})$ according to a local probabilistic model $P(r|r^{cur})$ that is monotonically non-increasing with the function value of $F_p(r; \mu)$. This probabilistic solution generation mechanism is also called a *probabilistic improvement navigation strategy*. The probabilistic improvement navigation strategy can be incorporated into IPC provided that $\forall i, j \in \{1, \dots, n\}$ and $j \neq i$, $\exists x_i \in \mathbf{R}^+$ and $y_{i,j} \in \mathbf{Z}^+$ such that the function value of $F_p(r; \mu)$ for $r \in N_i(r^{cur})$ is monotonically non-decreasing with the partial function $F_{q,i}(r; \mu, x, y)$ given in (27). As such, given an initial solution r^{init} , the stochastic local search algorithm iteratively moves to a neighboring solution by probabilistically generating a solution from $N_i(r^{cur})$ according to a local probabilistic model $P(r|r^{cur})$ that is monotonically non-increasing with the function value of $F_{q,i}(r; \mu, x, y)$ ($\forall i, j \in \{1, \dots, n\}$ and $j \neq i$, $x_i \in \mathbf{R}^+$, $y_{i,j} \in \mathbf{Z}^+$). This algorithm terminates when a preset time bound is reached. Other termination

criteria can also be employed, which will be discussed when solving the specific MAPP problems. The proposed stochastic local search algorithm is called the *Probabilistic Iterative Path Coordination* (PIPC) algorithm. In the following sections, the local probabilistic model $P(r|r^{cur})$ is developed.

5.2.1 Potential Function

A path r_i is considered successfully generated when all its path nodes have been created. Since each path r_i consists of a sequence of T_i path nodes $(r_i(0), r_i(1), \dots, r_i(T_i))$ and the type of each path node $r_i(t)$ only depends on its previous path node $r_i(t-1)$, and the incremental paths of other agents $r_{-i}^{cur} = (r_1^{cur}, r_2^{cur}, \dots, r_{i-1}^{cur}, r_{i+1}^{cur}, \dots, r_n^{cur})$, the success probability $P_s(r_i|r_{-i}^{cur})$ for each path r_i with respect to the other paths r_{-i}^{cur} could be defined as

$$P_s(r_i|r_{-i}^{cur}) = P_s(r_i(0)|r_{-i}^{cur}) \prod_{t=1}^{T_i} P_s(r_i(t)|r_i(t-1), r_{-i}^{cur}) \quad (58)$$

In order to handle this probability assignment, two parameters are introduced to define the success probability of the free nodes and collision nodes in each path r_i . The first parameter is called the minimization parameter η and it controls the success probability $P_s(r_i(t)|r_i(t) = v)$ of a free node $r_i(t) = v$, which is given by

$$P_s(r_i(t)|r_i(t) = v) = \eta_i, \text{ where } \eta_i \in [0,1] \quad (59)$$

In short, a free node $r_i(t)$ is generated successfully with the probability η_i . With $\eta_i=0$, it is never generated and with $\eta_i=1$, it is always generated.

The second parameter is the relaxation parameter β , which controls the probability that a collision node $r_i(t)=\bar{v}$ can be relaxed to be like a free node such that the path node $r_i(t)$ can be further generated with the probability η_i . When $\beta=0$, a collision node has no chance of

being treated like a free node, while a value of $\beta=1$, it can be considered to be exactly like a free node. Since the collisions in a collision node may be caused by one or multiple agents, so the probability of violating a dynamic constraint associated with agent A_j ($j=1,2,\dots,n, j \neq i$) is equal to $\beta_{i,j}$, then the relaxation probability of the collision node $r_i(t) = \bar{v}$ is given by

$$P_r(r_i(t)|r_i(t) = \bar{v}) = \prod_{j=1, j \neq i}^n (\beta_{i,j})^{D_{i,j}(t)} \quad (60)$$

where $\beta_{i,j} \in [0,1]$ and $D_{i,j}(t)$ indicates whether there is a collision between agent A_i and A_j at the time step t , $D_{i,j}(t)=1$ if it exists, $D_{i,j}(t)=0$ otherwise. The success probability $P_s(r_i(t)|r_i(t) = \bar{v})$ of the collision node $r_i(t)$ is then given by

$$P_s(r_i(t)|r_i(t) = \bar{v}) = \eta_i * \prod_{j=1, j \neq i}^n (\beta_{i,j})^{D_{i,j}(t)} \quad (61)$$

where $\eta_i \in [0,1]$ and $\beta_{i,j} \in [0,1]$. Next, the success probability of a path r_i with respect to the paths of other $(n-1)$ agents given the success probabilities for two types of path nodes is defined. If the path node $r_i(t)$ where $t=1,2,\dots,T_i$ is a free node v , then the success probability

$$P_s(r_i(t)|r_i(t-1), r_i(t) = v, r_{-i}^{cur}) = P_s(r_i(t)|r_i(t)=v) \quad (62)$$

If the path node $r_i(t)$ where $t=1,2,\dots,T_i$ is a collision node $\bar{v}$, then

$$P_s(r_i(t)|r_i(t-1), r_i(t) = \bar{v}, r_{-i}^{cur}) = P_s(r_i(t)|r_i(t)=\bar{v}) \quad (63)$$

Since the path node $r_i(0)$ denotes the starting position of the agent A_i and is therefore already generated, so $P_s(r_i(0)|r_{-i}^{cur})=1$. Then the success probability $P_s(r_i|r_{-i}^{cur})$ of a path r_i with respect to the other paths $r_{-i}^{cur} = (r_1^{cur}, r_2^{cur}, \dots, r_{i-1}^{cur}, r_{i+1}^{cur}, \dots, r_n^{cur})$ is given by

$$P_s(r_i|r_{-i}^{cur}) = P_s(r_i(0)|r_{-i}^{cur}) \prod_{t=1}^{T_i} P_s(r_i(t)|r_i(t-1), r_{-i}^{cur}) \quad (64)$$

$$= \eta_i^{f_i(r)} \prod_{j=1, j \neq i}^n (\beta_{i,j})^{\Gamma_{i,j}(r)}$$

where $r_i \in R_i(r^{cur})$, $\eta_i \in [0,1]$, $\beta_{i,j} \in [0,1]$, $\Gamma_{i,j}(r)$ given in (26) measures the violations of the dynamic constraints (2b) between the paths r_i and r_j ($i \in \{1,2,\dots,n\}$, $j \in \{1,2,\dots,n\}$ and $j \neq i$).

5.2.2 Local Probabilistic Model

Assume $R_i(r^{cur})$ includes all possible path assignments for r_i in $N(r^{cur})$, the conditional probability distribution $P(r_i|r_{-i}^{cur})$ can be obtained by normalizing the potential function $P_s(r_i|r_{-i}^{cur})$ in (64). However, since the probability assignment $P_s(r_i|r_{-i}^{cur})$ for each path r_i in $R_i(r^{cur})$ is unknown before it is evaluated and the size of $R_i(r^{cur})$ can be very large. As such, it is difficult to apply techniques like rejection sampling [108] or slice sampling [109]. A naïve idea is to generate each possible path r_i in $R_i(r^{cur})$ according to $P_s(r_i|r_{-i}^{cur})$ independently and then select the best one. Unfortunately, this is computationally expensive. The computational cost can be reduced by generating the possible paths in $R_i(r^{cur})$ with $P_s(r_i|r_{-i}^{cur})$ in an ascending order based on the value of the partial function $F_{q,i}(r; \mu, x, y)$ in (27). In so doing, the path generation procedure can be immediately terminated once a path is successfully generated. This path generation procedure can be modeled using a geometric distribution, in which a sequence of independent trials is run until the first success is obtained. These independent trials can be Bernoulli trials, in which the random variable x has two values, taking 1 with success probability p and 0 with failure probability $1 - p$, as given by

$$P_b(x) = \begin{cases} p, & \text{if } x = 1 \\ 1 - p, & \text{if } x = 0 \end{cases} \quad (65)$$

If the generation of each path r_i is considered a Bernoulli trial with the success probability $p = P_s(r_i|r_{-i}^{cur})$, then the geometric probability distribution $P_g(r_i^{(m)}|r_{-i}^{cur})$ that the m th trial is the first successful one is given by

$$P_g(r_i^{(m)}|r_{-i}^{cur}) = P_s(r_i^{(m)}|r_{-i}^{cur}) \prod_{l=1}^{m-1} (1 - P_s(r_i^{(l)}|r_{-i}^{cur})), \text{ where } m=1,2,3,\dots \quad (66)$$

Unfortunately, the value of $F_{q,i}(r; \mu, x, y)$ for each possible path in $R_i(r^{cur})$ is unknown a priori. It will be tricky to determine the trial orders of each possible path in $R_i(r^{cur})$. This problem can be addressed by adopting the lowest cost first algorithm, in which the path nodes of each possible path in $R_i(r^{cur})$ are evaluated in the partial order of $F_{q,i}(r; \mu, x, y)$. In this way, the possible path in $R_i(r^{cur})$ with lower $F_{q,i}(r; \mu, x, y)$ values can be generated first before others with higher $F_{q,i}(r; \mu, x, y)$ values. However, it is inevitable that a partial part of other possible paths must still be evaluated. Given the very large size of $R_i(r^{cur})$, the computational cost will still be high, especially when $P_s(r_i|r_{-i}^{cur})$ is too small for each path in $R_i(r^{cur})$, thus leading to many rejections. Memory and computational requirements can be significantly reduced by using a branch and cut technique to eliminate the evaluation of repeated path nodes. This can be realized by using a Space-Time map. To further improve computational efficiency, the A* algorithm is employed to generate a lowest possible cost path in $R_i(r^{cur})$ with respect to the function $F_{q,i}(r; \mu, x, y)$. The pseudo code of the probabilistic path planner is described in Figure 5-1.

Algorithm 5-1: ProbabilisticPathPlanner ($start, goal, lb_i$)

```

1)  $U \leftarrow \emptyset, Q \leftarrow \emptyset;$ 
2)  $Q \leftarrow Q \cup start;$ 
3) While  $Q \neq \emptyset$ 
4)    $v \leftarrow \operatorname{argmin}_{v \in Q} f(v);$ 
5)    $Q \leftarrow Q \setminus v;$ 
6)    $U \leftarrow U \cup v;$ 
7)   If  $v=goal$  and  $t_v > lb_i$ , return true;
8)    $\forall a \in \{N, W, E, S, I\},$ 
9)      $v' \leftarrow \operatorname{successor}(v, a);$ 
10)    If  $v' \notin U$ 
11)      If  $v' \notin Q$  and  $\operatorname{success}(v')$ 
12)         $Q \leftarrow Q \cup v';$ 
13)         $\operatorname{parent}(v') \leftarrow v;$ 
14)         $g(v') \leftarrow g(v) + c(v, v');$ 
15)         $f(v') \leftarrow g(v') + h(v');$ 
16)      Else if  $v' \in Q$  and  $g(v') > g(v) + c(v, v')$  and  $\operatorname{success}(v')$ 
17)         $\operatorname{parent}(v') \leftarrow v;$ 
18)         $g(v') \leftarrow g(v) + c(v, v');$ 
19)         $f(v') \leftarrow g(v') + h(v');$ 
20)      Else continue;
21) return false;
    
```

Figure 5-1. Probabilistic Path Planner in the Space-Time map.

The input $start$ and $goal$ denotes the starting position and target position of agent A_i , and lb_i is the lower bound on T_i that guarantees no collision exists after lb_i , and it is set to the last time step when other agents occupy the destination of the agent A_i before $T_{-i} = \max_{1 \leq j \leq n, j \neq i} (T_j)$. The A* algorithm begins with empty lists U and Q before the starting node $start$ is added to Q . At each iteration, the node $v = (x_v, y_v, t_v)$ with the least $f(v)$ is fetched out from Q in line 4. The path planning outer loop exits when $x_v = x_{goal}, y_v = y_{goal}$,

and $t_v > lb_i$. If not, the search continues. For each successor v' that is obtained by taking an action $a \in \{N, W, E, S, I\}$ at the node v (line 8-9), if v' has not been explored ($v' \notin U \cup Q$), or explored ($v' \in Q$) but with higher $g(v')$, then the node v' is accepted with success probability $P_s(r_i(t)|r_i(t-1), r_i^{cur})$. If accepted, it is added into list Q if it has not been explored. Its $g(v')$ is then set equal to $g(v)$ of its parent node v plus an extra cost c . Then the cost $f(v')$ is the sum of $g(v')$ and a heuristic function $h(v')$. The algorithm returns *false* if Q is empty.

Thus, the local probabilistic model with hyper-parameters η and β is given by

$$P_{\eta, \beta}(r|r^{cur}) = P(r_i|r_i^{cur}) = \frac{1}{Z} * P_g(r_i|r_i^{cur}) \text{ for each solution } r \in N_i(r^{cur}) \quad (67)$$

where Z is a normalized constant and it is given by

$$Z = \sum_{r_i \in R_i(r^{cur})} P_g(r_i|r_i^{cur}) \quad (68)$$

5.2.3 Parameter Tuning Strategies for Local Probabilistic Model

To encourage the convergence of PIPC to a locally optimal solution, the probability assignment $P_{\eta, \beta}(r|r^{cur})$ for each solution r in $N_i(r^{cur})$ should be non-increasing with the value of $F_{q,i}(r; \mu, x, y)$ in (27). As such, the performance of the PIPC algorithms is highly dependent on the parameters within $P_{\eta, \beta}(r|r^{cur})$, namely the minimization parameter η and the relaxation parameter β . This section will introduce different parameter setting strategies for these two parameters.

Condition 1. $\forall i=1,2,\dots,n$, $\eta_i = \eta^{x_i}$, $\beta_{i,j} = \beta_i^{y_{i,j}}$, $\beta_i \leq \eta^\mu$ and $\mu=M$, where $j=1,2,\dots,n$, $i \neq j$, $x \in \mathbf{R}^+$, $y \in \mathbf{Z}^+$, M is a sufficiently high value such that $M > x_i * f_i(r)$ for each solution $r \in N_i(r^{cur})$, $\eta \in (0,1)$ and $\beta_i \in (0,1)$.

Theorem 1. If **Condition 1** holds, $P_{\eta,\beta}(r|r^{cur})$ in (67) for $r \in N_i(r^{cur})$ is strictly decreasing with $F_{q,i}(r; \mu, x, y)$ in (27). (See proof of **Theorem 1** in appendix A.2)

The parameter setting **Condition 1** is theoretically valid but is not computationally efficient in practice. Firstly, random search is often costly since it does not guarantee continuous improvement of the solution. Secondly, having a solution selection mechanism that is biased toward improving solution will favor an intensive search around the local optima, thus preventing an extensive search that may produce even better solutions. The first problem can be ameliorated by incorporating a deterministic greedy search that can speed up the convergence rate of the algorithm to a locally optimal solution. This can be realized by setting integer values for η and β . In so doing, the condition 1 is relaxed to the following condition:

Condition 2. $\forall i=1,2,\dots,n$, $\eta_i=\eta^{x_i}$, $\beta_{i,j}=\beta_i^{y_{i,j}}$, $\beta_i \leq \eta^\mu$ and $\mu=M$, where $j=1,2,\dots,n$, $i \neq j$, $x \in \mathbf{R}^+$, $y \in \mathbf{Z}^+$ and M is a sufficiently high value such that $M > x_i * f_i(r)$ for each solution $r \in N_i(r^{cur})$. Unlike **Condition 1**, $\eta \in (0,1]$ and $\beta_i \in [0,1]$ for this condition.

Theorem 2. If **Condition 2** holds, $P_{\eta,\beta}(r|r^{cur})$ in (67) for $r \in N_i(r^{cur})$ is monotonically non-increasing with $F_{q,i}(r; \mu, x, y)$ in (27). (See proof of **Theorem 2** in appendix A.2)

Under **Condition 2**, the monotonic relationship between $P_{\eta,\beta}(r|r^{cur})$ and $F_{q,i}(r; \mu, x, y)$ will still hold even though it is not strictly decreasing like that under **Condition 1**.

To address the second problem, an extensive search should be favored when further improvement in the current intensive search is not forthcoming. Finding good solutions within a limited computational time requires one to constantly switch between intensive and

extensive search when appropriate conditions have been detected. Adaptive parameter setting strategies for β and η are proposed to achieve this feature in the proposed algorithm.

Adaptive β strategy - The relaxation parameter β only influences the success probability of infeasible MAPP solutions. A high value of β slows down the convergence of the stochastic local search algorithm to a feasible solution while a low value leads to a high probability of rejections in a crowded environment. As a result, finding the ideal value of β at each iteration of PIPC can be very tricky. To address this issue, an incremental β strategy is proposed. In this strategy, β is initially set to 0 and then gradually increased by an interval Δ_β until a path is successfully generated or an upper bound β_{max} is reached, where $\beta_{max} = \eta^\mu$ according to **Condition 2**. The value of β is allowed to increase gradually from 0, as this results in a lower search cost for infeasible solutions compared to the alternative of decreasing β gradually from 1.

Adaptive η strategy - The minimization parameter η controls the success probability of all the solutions in $N_i(r^{cur})$. Setting η to a low value leads to a high probability of rejections at each iteration of PIPC, thus incurring a higher computational cost. On the other hand, a high value of η favors the selection of the solution r with the lowest $F_{q,i}(r; \mu, x, y)$ in $N_i(r^{cur})$ and will compromise the ability of the algorithm in escaping locally optimal basins, especially feasible locally optimal basins, thus reducing the chance of finding a higher quality solution. As a result, a useful strategy is to always set η to a high value of 1 but a low value when it is detected that PIPC is entering a feasible region. To avoid producing too many rejections due to a low value of η , a lower bound η_{min} for η is imposed. In order to determine a suitable value for η_{min} , consider at each iteration τ the success probability of a solution r associated

with η is $\eta^{x_i * f_i(r)}$ ($i \in \{1, \dots, n\}$). In order to avoid too many rejections, $\eta^{x_i * f_i(r)}$ of the current solution r^{cur} should not be too small. As such, a constraint is imposed for η and is given by

$$\eta^{x_i * f_i(r^{cur})} \geq \varepsilon_1 \quad (69)$$

where $\varepsilon_1 \in [0, 1]$. The function (69) implies $\eta \geq \frac{x_i * f_i(r^{cur})}{\sqrt{\varepsilon_1}}$. Similarly, a constraint is imposed on $\beta_{i,j}$ such that it is not forced to be a small value. As $\beta_{i,j}$ ($i \in \{1, \dots, n\}, j \in \{1, \dots, n\}$ and $i \neq j$) can be different, in this case, the constraint is given by

$$\beta_i^{\overline{y_{i,j}}} \geq \varepsilon_2 \quad (70)$$

where $\overline{y_{i,j}}$ is the mean value of $y_{i,j}$, $\varepsilon_2 \in [0, 1]$. As $\beta_i \leq \eta^\mu$, so $\eta \geq \frac{\mu * \overline{y_{i,j}}}{\sqrt{\varepsilon_2}}$. Thus, the minimum value of the minimization parameter η is given by

$$\eta_{min} = \max(\frac{x_i * f_i(r^{cur})}{\sqrt{\varepsilon_1}}, \frac{\mu * \overline{y_{i,j}}}{\sqrt{\varepsilon_2}}) \quad (71)$$

Since the values of ε_1 and ε_2 defines the lower bound on the success probability of a feasible solution and the relaxation probability of a collision node respectively, values of ε_1 and ε_2 that are too low could lead to a very low success probability of a solution and therefore results in many rejections, while high values of ε_1 and ε_2 prevent η_{min} from being set to a sufficiently low value so as to allow the algorithm to escape from feasible locally optimal basins. The values of $\varepsilon_1=0.1$ and $\varepsilon_2=0.1$ were empirically found to be a suitable compromise.

The pseudo-code of PIPC using the local probabilistic model $P_{\eta,\beta}(r|r^{cur})$ for MAPP is described in Figure 5-2. Then function **ProbabilisticPathPlanner** ($N_i(r^{[\tau-1]})$, $P_{\eta,\beta}(r|r^{[\tau-1]})$) generates a neighboring solution from the local neighborhood $N_i(r^{[\tau-1]})$ by the probabilistic path planner given in Figure 5-1 with respect to the partial function $F_{q,i}(r; \mu, x, y)$ given in (27) and the local probabilistic model $P_{\eta,\beta}(r|r^{[\tau-1]})$ in (67) satisfying **Condition 2**. The

scheme *betaUpdate*() can either use the fixed parameter setting strategy that sets a fixed value for β , or the proposed adaptive parameter setting strategy in this section. The scheme *pcUpdate*() updates the value of μ using the adaptive μ strategy in section 3.2.2 with the penalty cost given in (29).

Algorithm 5-2: PIPC($r^{init}, r^{best}, P_{\eta, \beta}(r|r^{cur})$)

- 1) $r^{[0]} \leftarrow r^{init}$;
 - 2) $i \leftarrow 1, \tau \leftarrow 1, \eta \leftarrow 1$;
 - 3) Repeat the following procedure until some stopping criteria are satisfied:
 - a) $\mu \leftarrow pcUpdate()$;
 - b) At the iteration τ , repeat the following procedure until the iterative solution $r^{[\tau]}$ is successfully generated:
 - I. $\beta \leftarrow betaUpdate()$;
 - II. $r^{[\tau]} \leftarrow ProbabilisticPathPlanner(N_i(r^{[\tau-1]}), P_{\eta, \beta}(r|r^{[\tau-1]}))$;
 - c) If $r^{[\tau]}$ is feasible:
 - i. $\eta \leftarrow \eta_{min}$;
 - ii. If $F(r^{best}) > F(r^{[\tau]})$ or $r^{best} = \emptyset$, then $r^{best} \leftarrow r^{[\tau]}$;
 - d) If $r^{[\tau]}$ is not feasible, $\eta \leftarrow 1$;
 - e) $\tau \leftarrow \tau + 1$;
 - f) $i \leftarrow i + 1$ if $i < n$, otherwise $i \leftarrow 1$;
 - 4) Return $r^{[\tau-1]}$;
-

Figure 5-2. Pseudocode for the Probabilistic Iterative Path Coordination (PIPC) algorithm for MAPP.

5.3 Solving MAPP Optimization Problems using PIPC

5.3.1 Probabilistic Iterative Makespan Minimization for Makespan Minimized MAPP

Figure 5-3. Overview description of the Probabilistic Iterative Makespan Minimization (PIMM) algorithm for solving makespan minimized MAPP.

The iterative algorithm for solving makespan minimized MAPP using PIPC is shown in Figure 5-3. Beginning with an initial solution $r^{(0)}$, which is given by r^{init} , the iterative algorithm approximates the optimal solution of the makespan minimized MAPP problem by solving a sequence of minimization problem in (36) with the value of the control parameter c gradually increasing from 0 based on an increment rate of ε_c . At each iteration k , the minimization problem in (36) can be solved by $\mathbf{PIPC}(r^{(k-1)}, r^{best}, P_{\eta, \beta}(r|r^{cur}))$ given in Figure 5-2 provided that the problem specific cost function $F_p(r; \mu)$ given in (37) is monotonically non-decreasing with the function value of $F_{q,i}(r; \mu, x, y)$ in (27). This

assumption holds when the partial objective function $f_i(r)$ and the general parameters x and y in (27) are set as

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Phi_i(r) \quad (72)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1 / \sum_{j=1}^n \exp(c * [\Phi_j(r^{(k-1)}) - \Phi_i(r^{(k-1)})]) \}, \quad (73)$$

$$y_{i,j} = 1$$

where c is the control parameter in (32). The problem specific function $\Phi_i(r)$ given in (3) calculates the total time related cost of all unit actions in the path r_i . As a result, at each iteration k , the minimization problem in (36) is solved by **PIPC**($r^{(k-1)}, r^{best}, P_{\eta,\beta}(r|r^{cur})$) with the local probabilistic model $P_{\eta,\beta}(r|r^{cur})$ satisfying **Condition 2** under the settings in (72) and (73). The iterative algorithm using PIPC for makespan minimized MAPP is called Probabilistic Iterative Makespan Minimization (PIMM). Due to the nondeterministic nature of PIPC, PIMM is made to terminate when the current best solution has not been improved after δ iterations within PIPC or a preset time bound is reached.

5.3.2 Probabilistic Iterative Cumulation Minimization for Total Cost Minimized MAPP

The iterative algorithm for solving the total cost minimized MAPP problem using PIPC is shown in Figure 5-4. Beginning with the initial solution $r^{(0)}$, which is given by r^{init} , the optimal solution of the total energy cost minimized MAPP problem is approximated by iteratively solving the minimization problem in (42).

Figure 5-4. Overview description of the Probabilistic Iterative Cumulation Minimization (PICM) algorithm for solving total cost minimized MAPP.

At each iteration k , the minimization problem in (42) can be solved by $\mathbf{PIPC}(r^{(k-1)}, r^{best}, P_{\eta,\beta}(r|r^{cur}))$ in Figure 5-2, provided that the function value of $F_{q,i}(r; \mu, x, y)$ in (27) for $r \in N_i(r^{cur})$ is monotonically non-decreasing with the problem specific cost function $F_p(r; \mu)$ given in (43). This assumption holds when the settings of the partial objective function $f_i(r)$ and the general parameters x and y in (27) are given by

$$\forall i \in \{1, \dots, n\}, f_i(r) = \Psi_i(r) + \rho_c * I_i(r) \quad (74)$$

$$\forall i, j \in \{1, \dots, n\} \text{ and } j \neq i, x_i = 1, y_{i,j} = 1 \quad (75)$$

where ρ_c is the cost ratio in (41). The problem specific function $\Psi_i(r)$ given in (4) calculates the total distances traversed by the agent A_i . The problem specific function $I_i(r)$ given in (40) calculates the total cost of all unit actions in the path r_i . As summarized in Figure 5-4, at each iteration k , the minimization problem in (42) is solved by $\mathbf{PIPC}(r^{(k-1)}, r^{best}, P_{\eta,\beta}(r|r^{cur}))$ with the probabilistic model $P_{\eta,\beta}(r|r^{cur})$ satisfying **Condition 2** under the settings in (74)

and (75). This iterative algorithm using PIPC for total cost minimized MAPP is called Probabilistic Iterative Cumulation Minimization (PICM). Like PIMM, PICM can be made to terminate when the current best solution has not been improved after δ iterations within PIPC or a preset time bound is reached.

5.4 Experimental Evaluation

Two groups of experiments are presented in this section. The first group investigates the best strategy for setting performance-sensitive parameters like η and β for the PIPC algorithm. The second group of experiments investigates the performance of the proposed probabilistic PIPC-based algorithms when compared to the GIPC-based algorithms that employed guided local search. These comparisons were done under different size of traversable space by first varying the map size and then the agent count.

5.4.1 Evaluation of the PIPC Algorithm

The performance of the PIPC algorithm is highly dependent on the chosen values of the penalty coefficient μ , relaxation parameter β and minimization parameter η . The adaptive μ strategy described in section 3.2.2 has already been demonstrated to be a good method for determining the value of μ (see experimental results in section 3.4.1.1). Two different strategies for setting the values of η and β have been suggested in section 5.2.3, namely the fixed and adaptive parameter setting strategies. The following experiments evaluate if the adaptive parameter setting strategies of η and β can yield superior performances over fixed parameter setting strategies for the PIPC algorithm. These evaluations were done using the PIMM algorithm for solving the makespan minimized MAPP problem. Since the PIMM algorithm is an iterative algorithm that can improve the solution quality with increasing

computational time, the evaluations were done against allowable computational time. A 30x20 grid map (see Figure 3-4) with a higher agent count of 200 and static obstacle coverage of 20% was used in all the experiments.

All the evaluations in this section were done with the parameter settings of adaptive μ ($\alpha_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for PIPC and $\varepsilon_c=1$ for control parameter c in PIMM.

5.4.1.1 Evaluation of PIPC with different β parameter setting strategies with increasing allowable computational time

The following experiments evaluate the fixed and adaptive β strategies in influencing the performance of PIPC when used within PIMM for solving makespan minimized MAPP. Three different fixed values of β were chosen for the fixed β strategy, namely a low value of $\beta=0.1$, an intermediate value of $\beta=0.5$ and a high value of $\beta=0.8$. The adaptive β strategy gradually increases the value of β from 0 to η^μ by an interval $\Delta_\beta (0 \leq \Delta_\beta \leq \eta^\mu)$ until a new solution r' is successfully obtained at each iteration of PIPC. Three different increment rates Δ_β for the adaptive β strategy were evaluated. They were a low rate of $\Delta_\beta=0.05$, an intermediate rate of $\Delta_\beta=0.2$ and a high rate of $\Delta_\beta=0.5$. For these experiments, the minimization parameter η was set to $\eta=\eta_{min}$, as given in (71).

Figure 5-5. Success rate of PIMM with different β parameter setting strategies against the allowable computational time.

A high β increases the probability that infeasible MAPP solutions are accepted during each iteration of PIPC, which means more iterations are required to find a feasible solution. As a result, for a given increase in allowable computational cost, the high fixed β ($\beta=0.8$) leads to a correspondingly slower increase in success rate compared to the intermediate fixed β ($\beta=0.5$) (see Figure 5-5). On the other hand, if the value of β is fixed too low, the generation probability of infeasible solutions will be reduced and this will therefore increase the probability of rejections when searching in the infeasible solution regions. The result of this is a higher computational cost in generating an improving neighboring MAPP solution. As can be observed in Figure 5-5, the low fixed value of $\beta=0.1$ resulted in a slower increase in success rate compared to the intermediate fixed value of $\beta=0.5$. This competing effect on the computational behavior of PIPC suggests that a fixed value of β is tricky to determine and it should not be set either too high or too low.

Figure 5-6. Task completion time of PIMM with different β parameter setting strategies against the allowable computational time.

In addition, a low value of β leads to a more biased search towards feasible solutions, leading to an intensive search around feasible solution regions in the solution space. This implies that a superior success rate performance will result if enough computational time is provided. In contrast, a high value of β increases the probability of extensive search in the solution space and therefore results in a better task completion time performance (see Figure 5-6). In other words, there is an optimal choice for β that will strike the best tradeoff between the success rate and task completion time performance. This unfortunate tradeoff also suggests that adopting a fixed β value creates a dilemma that prevents one from obtaining superior performance in both success rate and task completion time. Furthermore, this optimal choice was observed to be dependent on environment characteristics such as the agent count and the size of traversable space and is therefore difficult to be determined in practice.

The adaptive β strategy attempts to address this tradeoff dilemma by finding the ideal value of β at each iteration of PIPC using a systematic procedure that progressively increases the value of β . The proposed adaptive β strategy was observed to have a faster increase in

success rate with allowable computational time than any of the fixed β values used (see Figure 5-5). Moreover, the adaptive β strategy is also able to produce a task completion time performance that is superior to even the high fixed $\beta=0.8$ value (see Figure 5-6).

The value of the interval Δ_β used in the adaptive β strategy also influences PIPC's performance. Finding an ideal value for β gives the best success rate performance for a given allowable computational cost. As such, a small Δ_β (e.g. $\Delta_\beta=0.05$) allows the higher resolution incrementing strategy to reach this ideal value of β in a more accurate manner, thus leading to better success rate performance (see Figure 5-5). However, a larger interval Δ_β (e.g. $\Delta_\beta=0.5$) often leads to a higher over-estimation of β , which actually results in a superior task completion time performance. The best compromise is the intermediate value of $\Delta_\beta=0.2$, which is able to achieve comparable success rate performance with the low $\Delta_\beta=0.05$ while maintaining the superior task completion time performance of the high $\Delta_\beta=0.5$.

5.4.1.2 Evaluation of PIPC with different η parameter setting strategies with increasing allowable computational time

Adopting the adaptive β strategy presented in the previous section, the next set of experiments investigates if an adaptive η strategy in PIPC can yield performance improvements in PIMM. A highest value of $\eta=1.0$ and a lowest value of $\eta=\eta_{min}$ were chosen for the fixed η strategy while the adaptive η strategy dynamically switches its value between 1.0 and η_{min} .

Figure 5-7. Success rate of PIMM with different η parameter setting strategies against the allowable computational time.

Figure 5-8. Task completion time of PIMM with different η parameter setting strategies against the allowable computational time.

A low value of η leads to more rejections at each iteration of PIPC and therefore results in a slower increase of success rate with the increasing allowable computational time (see Figure 5-7). However, PIPC with a low η value is more likely to find a superior makespan-minimized MAPP solution and therefore $\eta = \eta_{min}$ produces a lower task completion time for a

given high enough computational time (see Figure 5-8). This is because an increased probability of local convergence in PIPC due to a high value of η comes with the increased likelihood of the search process being trapped around feasible locally optimal basins. By contrast, adaptive η changes its value to η_{min} as PIPC searches around a feasible locally optimal solution, but it immediately reverts its value back to 1 once PIPC escapes from the trap. In this way, the adaptive η strategy can benefit from the fast feasible solution convergence rate associated with a high η and the superior performance of finding a makespan-minimized MAPP solution associated with a low η . The intelligent switching of the value of η in the adaptive η strategy also allows PIMM to find a superior MAPP solution using less computational time (see Figure 5-8).

In summary, fixed value strategies for η and β suffers from a tradeoff between intensive and extensive search in the MAPP solution space. In contrast, the proposed adaptive parameter setting strategies allow the values of η and β to be dynamically changed by some heuristics that can determine when it is most appropriate to adopt either an extensive or intensive search. It has been shown that this informed search in the MAPP solution space led to improved success rate and task completion time performances for a given allowable computational time. It has to be admitted that the proposed adaptive parameter tuning strategies for η and β are still sensitive to the environment since the value settings of the newly introduced parameters such as the increment rate Δ_β and some threshold values of such as ε_1 in (69) and ε_2 in (70) in the design of these strategies are empirically determined. Fortunately, compared with the main parameters η and β , the newly introduced parameters are less critical to the performance of the proposed PIPC algorithm.

5.4.2 Comparative Evaluation of PIPC-based Algorithms under Varying Map Size

The section compares the performance of PIPC with GIPC under different size of traversable space by varying map size. Like section 4.4.2.2, the comparison between PIPC and GIPC was only done when used for solving the makespan minimize MAPP problem. Two different sized grid maps, namely the small sized 20x15 grid map and the large sized 40x30 grid map, were used. The comparisons were done under the same fixed agent count of 160 against increasing iterations.

Figure 5-9. Comparative success rate of PIMM and GIMM with different map sizes against increasing iterations.

Figure 5-10. Comparative computational time of PIMM and GIMM with different map sizes against increasing iterations.

Compared with the large 40x30 grid map, the small 20x15 grid map has a higher map density. As a result, more iterations were used for GIMM and PIMM to find a feasible solution in the smaller 20x15 grid map (see Figure 5-9). Compared with GIPC, an improving neighboring solution in PIPC can be accepted with the probability $P_{\eta,\beta}(r|r^{cur})$ in (67). As the probability of accepting feasible MAPP solutions is always 1 while the probability of accepting infeasible solutions is less than 1 due to the relaxation parameter β when PIPC searches within the infeasible solution region, PIPC has a more biased search towards feasible MAPP solutions compared with GIPC. It was observed in Figure 5-9 that PIMM has a faster convergence rate towards feasible solutions than GIMM.

Unlike GIPC, PIPC evaluates the local neighborhood in a probabilistic manner and therefore has a lower computational cost in searching for an improving neighboring MAPP solution. Under such circumstances, it was observed that the computational time of PIMM increases slower than that of GIMM as the number of iterations increases (see Figure 5-10). Moreover, the computational efficiency of PIMM over GIMM is even more significant as the size of

traversable space increases. The results observed in Figures 5-9 and 5-10 do suggest that the design objective of improving GIPC's weakness of rapidly increasing computational time with increasing map density and size has been achieved with PIPC's stochastic-search technique.

Figure 5-11. Comparative task completion time of PIMM and GIMM with different map sizes against increasing iterations.

Unfortunately for PIPC, due to its probabilistic behavior in searching for an improving neighboring solution at each iteration, PIMM has a slower convergence rate towards an optimal solution compared with GIMM. Without sufficient iterations, the task completion time performance of PIMM is inferior to that of GIMM (see Figure 5-11, when iterations are < 1000). However, if more iterations were allowed, the task completion time performance of PIMM will eventually approach that of GIMM.

5.4.3 Comparative study of PIPC-based Algorithms under Varying Agent Count

The following experiments compare the performance of the PIPC-based algorithms with the GIPC-based algorithms under varying agent count in a 4x4 grid map as shown in Figure 5-12.

In the 4x4 grid map, the agent count was varied from 0 to 15. For each run with a different agent count, the same 100 random instances that have feasible solutions were used. The small-sized 4x4 grid map was used because it allows the overall performance of the PIPC-based and GIPC-based algorithms to be compared in an environment that can vary from a sparsely crowded to an extremely crowded scenario. The small map size of 4x4 also facilitates experimental results to be obtained within a reasonable computational time.

Figure 5-12. A random scenario in a 4x4 grid map with 15 agents. The small grey squares denote the mobile agents and the dashed squares denote their respective unique destinations.

The comparisons between these MAPP algorithms were done under the same time bound of 5 minutes with the common parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for GIPC and PIPC, the increment rate $\varepsilon_c=1$ for control parameter c in GIMM and PIMM, the cost ratio $\rho_c=0.1$ for GICM and PICM. Other specific parameter settings of adaptive η , adaptive β with the interval $\Delta_\beta=0.2$ and an iteration bound of $\delta=\varepsilon_\delta*n$ (ε_δ is a constant value, n is the number of agents) on the improvement of the solution quality were used in PIPC. Two different values of $\varepsilon_\delta=1$ and $\varepsilon_\delta=15$ were used in the experiments presented in this chapter.

5.4.3.1 Comparative evaluation of PIMM for makespan minimized MAPP

The following experiments compare the performance of PIMM with GIMM in solving makespan minimized MAPP on a 4x4 grid map.

Figure 5-13. Comparative success rates of GIMM and PIMM under varying agent count in a 4x4 grid map.

Figure 5-14. Comparative task completion time of GIMM and PIMM under varying agent count in a 4x4 grid map.

Figure 5-15. Comparative computational time of GIMM and PIMM under varying agent count in a 4x4 grid map.

Due to a lower probability of accepting non-improving solutions, PIPC can only escape from locally optimal basins with a small probability and is therefore not efficient as GIPC in escaping from infeasible locally optimal basins. Moreover, the probabilistic nature of PIPC causes it to repeatedly re-visit locally optimal basins as it retains no information of previously visited locally optimal basins. As the environment gets crowded, the probability of re-visiting infeasible locally optimal basins will increase within PIPC. In contrast, GIPC steers the local search away from infeasible locally optimal basins by modifying the guided cost function such that the cost of the infeasible locally optimal solution in the basin is increased higher than surrounding solutions. In so doing, GIPC can also avoid re-visiting the infeasible locally optimal basins. As a result, due to PIPC's computationally efficiency in searching for an improving neighboring solution, PIMM ($\delta=n$) can be observed in Figure 5-15 to be still more computationally efficient than GIMM in finding a feasible MAPP solution in a sparsely and moderately crowded environment (agent count < 12). However, when the environment becomes highly crowded, the possibility that the local search enters infeasible

locally optimal basins rapidly increases. As a result, the computational performance of PIMM deteriorates quickly in a highly crowded environment (agent count > 12). PIMM's other performance measures such as success rate and task completion time are also observed to be significantly affected (see Figures 5-13 and 5-14). In the extremely crowded environment (agent count=15), it was observed that PIMM was unable to find a feasible solution within the time bound of 5 minutes (see Figure 5-13). In contrast, within the time bound of 5 minutes, GIMM is still able to maintain good success rate, task completion time and computational time performances as the agent count increases to 15 in the 4x4 grid map.

Notice in Figure 5-14 that the quality of the task completion time solutions of PIMM can be improved if the current best solution is allowed to stay non-improving for a larger number of δ iterations (i.e. $\delta=15*n$ compared to $\delta=n$). However, the unfortunate side-effect of these better quality solutions is an increase in computational cost (see Figure 5-15).

5.4.3.2 Comparative evaluation of PICM for total cost minimized MAPP

The following experiments compare the performance of PICM with GICM in solving total cost minimized MAPP on a 4x4 grid map.

Figure 5-16. Comparative success rate of GICM and PICM under varying agent count in a 4x4 grid map.

Figure 5-17. Comparative cumulative cost of GICM and PICM under varying agent count in a 4x4 grid map.

Figure 5-18. Comparative computational time of GICM and PICM under varying agent count in a 4x4 grid map.

Like the experimental results for the makespan minimized MAPP algorithms, similar comparative performances of GICM and PICM for solving total cost minimized MAPP were observed. PICM was observed to enjoy better computation time performance over GICM while maintaining comparable success rate and task completion performances until the environment became highly crowded (i.e. agent count > 12). On the other hand, GICM was observed to have superior performances in all categories when the environment was highly crowded.

5.5 Summary

This chapter first proposed a stochastic local search algorithm called Probabilistic Iterative Path Coordination (PIPC), which addresses IPC's problem of being trapped in a locally optimal basin by allowing non-improving solutions to be accepted with a certain probability. As shown in Figures 5-3 and 5-4, two PIPC-based iterative algorithms called Probabilistic Iterative Makespan Minimization (PIMM) and Probabilistic Iterative Cumulation Minimization (PICM) were then proposed by replacing IPC with PIPC in the iterative

frameworks for solving makespan minimized and total cost minimized MAPP in Figures 3-2 and 3-3. The PIPC-based algorithms were finally evaluated under different parameter settings and compared with GIPC-based algorithms under different environment characteristics. Compared to GIPC, PIPC has several innate strengths. Firstly, due to the design of its probabilistic solution generation mechanism, the search in PIPC is made more biased towards feasible MAPP solutions. This allows PIPC to exhibit superior success rate performance, especially if only limited computational time is allowed. Secondly, PIPC is more computationally efficient in finding a neighboring solution and is therefore more scalable to the size of traversable space than GIPC. In other words, under normal circumstances such as moderately crowded environments, PIPC is computationally much faster than GIPC in finding a feasible solution.

However, the weaknesses of PIPC become apparent as the environment gets highly crowded. In such a scenario, the probability of entering infeasible locally optimal basins quickly increases. Compared to GIPC, PIPC's lower probability of accepting worse solutions now works against it, making it inefficient in getting out of such locally optimal basins. This means PIPC's success rate performance in highly crowded scenarios deteriorates quickly, along with the increasing computational cost needed by PIPC to find feasible solutions.

In terms of application scenarios, PIPC's strength is best demonstrated in large sparsely and moderately crowded environments, while GIPC's strength is best demonstrated in small highly crowded environments. In the next chapter, a hybrid variant of IPC that combines the strengths of PIPC and GIPC is proposed.

6 ITERATIVE PATH COORDINATION USING HYBRID HEURISTIC

Though GIPC and PIPC share the same local search framework for MAPP, they exhibit different strengths and weaknesses due to the differences in their cost function formulation and navigation strategy. However, the weakness of one approach can be ameliorated if it is complemented by the strength of the other. In the following sections, a hybrid heuristic algorithm named Reactive Iterative Path Coordination (RIPC) for MAPP is proposed to explore the possibility of achieving this complementary benefit.

6.1 Reactive Iterative Path Coordination (RIPC)

GIPC is able to navigate the search out of infeasible locally optimal basins by modifying a guided cost function, but its search for an improving neighboring MAPP solution in a local neighborhood is computationally expensive due to the used best improvement navigation strategy. By contrast, PIPC saves computational cost in the search process by generating neighboring MAPP solutions probabilistically, but it is not computationally efficient in escaping from a locally optimal basin due to the lower probability of accepting non-improving solutions. The downsides of these two algorithms can be addressed by combining GIPC's guided cost function $F_g(r; \mu, h)$ in (48) and guided mechanism in section 4.2 with PIPC's probabilistic improvement navigation strategy in section 5.2 together in a local search algorithm. To achieve this, the generation probability $P_{\eta, \beta}(r | r^{cur})$ in (67) for each solution r in $N_i(r^{cur})$ should be monotonically non-increasing with the increasing value of the guided cost function $F_g(r; \mu, h)$. This is viable as the problem specific guided cost function $F_g(r; \mu, h)$ for makespan minimized and total cost minimized MAPP problems can be monotonically

non-decreasing with the function value of $F_{q,i}(r; \mu, x, y)$ in (27) for $i, j \in \{1, \dots, n\}$ and $j \neq i$, $x_i \in \mathbf{R}^+$, $y_{i,j} \in \mathbf{Z}^+$, as was described in section 4.3. Thus, at each iteration τ , the hybrid algorithm probabilistically generates a neighboring solution from the local neighborhood $N_i(r^{cur})$ according to the local probabilistic model $P_{\eta,\beta}(r|r^{cur})$ satisfying **Condition 2** in section 5.2.3. Whenever an infeasible locally optimal solution is encountered, the deterministic guided mechanism described in section 4.2 is used. In order to do this, one must have the ability to detect if the search is currently in a local optimum. Since a locally optimal solution is the best solution within its all local neighborhoods, a simple and useful method is to verify whether the current solution is the best within its all local neighborhoods. The simple way used in GIPC is to detect whether the current solution is continuously kept for at least n iterations, which is mathematically given by

$$\alpha \geq n \tag{76}$$

where n is the number of agents. The variable α records the actual number of times that the current solution continuously remains unchanged in the past n iterations. This detection condition in (76) works well when the best improvement navigation strategy is used. However, for the probabilistic improvement navigation strategy, it is not reliable as the probabilistic improving navigation strategy does not guarantee to find the best solution from each local neighborhood. To allow the combined techniques to work effectively, the best improvement navigation strategy is used each time the search gets trapped in an infeasible locally optimal basin such that an infeasible locally optimum can be reliably detected given the condition in (76). After escaping from this basin, the navigation strategy is immediately switched back to the probabilistic improvement one. To achieve this idea, two issues need to be addressed. One is to detect whether the current solution search is trapped in an infeasible locally optimal basin. The other is the scheme to switch between the best and probabilistic

improvement navigation strategies. To address these two issues, a probabilistic detection mechanism and a switching scheme are proposed in the following sections.

6.1.1 Probabilistic Detection Mechanism

Being a locally optimal solution, the solution r^* has the lowest path cost of r_i in each of its local neighborhood $N_i(r^*)$ ($i \in \{0, 1, \dots, n\}$). As the solution r in the local neighborhood $N_i(r)$ with the lower path cost of r_i has a higher generation probability according to the local probabilistic model $P_{\eta, \beta}(r|r^{cur})$ given in (67), so the locally optimal solution r^* has the highest generation probability than its surrounding solutions. In this case, once the solution search gets trapped in an infeasible locally optimal basin, the current solution has a higher chance to remain unchanged compared to other situations. Based on this intuition, it is possible to probabilistically decide that the current solution search is trapped in an infeasible locally optimal basin when the ratio of the iterations that the iterative solution remains unchanged to the total iterations is above a certain threshold. As such, assuming the infeasible iterative solutions in a total of π iterations of the hybrid algorithm are used for this detection, the value θ_π records the actual number of times that the iterative solution remains unchanged during the π iterations and its value can be directly obtained during the solution search process. On the other hand, the value θ_π is the expected number of times that the iterative solution remains unchanged during the π iterations and it can be estimated based on the success probability $P_s(r_i|r_i^{cur})$ in (64) of the iterative solution. Since η is always set to 1 in the adaptive η strategy when searching in infeasible regions, so the expected value of θ_π is calculated as

$$\theta_\pi = \sum_{\tau}^{\tau+\pi} \prod_{j=1, j \neq i}^n (\beta_{i,j})^{\Gamma_{i,j}(r^{[\tau]})} = \sum_{\tau}^{\tau+\pi} \beta_i^{\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{[\tau]})\}} \quad (77)$$

where $i \in \{1, 2, \dots, n\}$, $r^{[\tau]}$ is the infeasible iterative solution at the iteration τ of the hybrid algorithm. As a result, the current solution search is assumed to be trapped in an infeasible locally optimal basin when

$$\theta_\pi / \theta_\pi \geq p \quad (78)$$

where p is a possibility value between 0 and 1. Two ways can be used for setting the value of p . A simple way is to set the value of p to a fixed value. The alternative way is to set a heuristic value to p . The heuristic value can be designed by taking into account information related to how crowded the environment is. These could include the map density or the ratio of the size of the traversable space to the agent count. A proposed heuristic value of p used in the experiments is given by

$$p = \varepsilon_p * \rho_p^{b_p} \quad (79)$$

where ε_p is a constant value and b_p is an exponential factor. The variable ρ_p is the ratio of the number of traversable vertices in the search map to the agent count. The traversable vertices are essentially unoccupied nodes in the grid map. The design of the heuristic formulation in (79) is based on the intuition that the probability that the hybrid algorithm gets trapped in infeasible locally optimal basins increases as the environment gets crowded. In this case, the probability of switching to the best improvement navigation strategy should be higher in a more crowded environment such that the guided mechanism described in section 4.2 can be employed to efficiently navigate the search out of infeasible locally optimal basins. The suitable values for scaling rate ε_p and the exponential factor b_p in (79) were empirically determined to be 0.05 and 5 respectively.

When the solution search is probabilistically detected to be trapped in an infeasible locally optimal basin, the switch between the probabilistic and best improvement navigation strategy is realized by the switching scheme proposed in the next section.

6.1.2 Switching Scheme

Compared with the best improvement navigation strategy, the probabilistic improvement navigation strategy has an additional probabilistic mechanism that selects a neighboring solution from $N_i(r^{cur})$ probabilistically before it is evaluated. In this case, the best neighboring solution can only be obtained with the probability $P_{\eta,\beta}(r|r^{cur})$ in (67) in the probabilistic improvement navigation strategy. As a result, the probabilistic improving navigation strategy can be made as best improvement navigation strategy if the best or best feasible neighboring solution in $N_i(r^{cur})$ is obtained with probability 1. According to the local probability model $P_{\eta,\beta}(r|r^{cur})$ in (67), this can be realized by either setting the success probability $P_s(r_i|r_{-i}^{cur})=1$ for each $r_i \in R_i(r^{cur})$, or setting $P_s(r_i|r_{-i}^{cur})=1$ for viable paths in $R_i(r^{cur})$ and $P_s(r_i|r_{-i}^{cur})=0$ for other paths in $R_i(r^{cur})$. Since the success probability $P_s(r_i|r_{-i}^{cur})$ is controlled by the parameters η and β , and η is always set to 1 in the adaptive η strategy when searching in infeasible regions, then the probability assignment $P_s(r_i|r_{-i}^{cur})$ for $r_i \in R_i(r^{cur})$ is determined by the value of β . When β is set to 1, the probability assignment $P_s(r_i|r_{-i}^{cur})$ can be made to 1 for each $r_i \in R_i(r^{cur})$. Moreover, if β is set to 0, the probability assignment $P_s(r_i|r_{-i}^{cur})$ can be made to 1 for all viable paths in $R_i(r^{cur})$ and 0 for other paths in $R_i(r^{cur})$. The integer value setting for β can be done by setting the value of Δ_β to 1. Thus, the switch between the probabilistic and best improvement navigation is realized by setting the value of Δ_β to a real value ε_Δ ($0 < \varepsilon_\Delta < 1$) or an integer value 1.

As a result, each time the current solution search of the hybrid algorithm is probabilistically detected to be trapped in an infeasible locally optimal basin, the value of Δ_β is set to 1. Otherwise the value of Δ_β is set to a fixed value ε_Δ . As the condition in (78) is verified every π iterations, the value of Δ_β is updated every π iterations according to the function given by

$$\Delta_\beta = \begin{cases} 1, & \text{if } \theta_\pi / \Theta_\pi \geq p \\ \varepsilon_\Delta, & \text{otherwise} \end{cases} \quad (80)$$

In summary, beginning with an initial solution, at each iteration τ , the hybrid algorithm probabilistically generates a neighboring solution from the local neighborhood $N_i(r^{cur})$ according to the local probabilistic model $P_{\eta,\beta}(r|r^{cur})$ satisfying **Condition 2** in section 5.2.3. After every π iterations, the condition in (78) is verified and then the value of Δ_β is updated according to (80). Whenever an infeasible locally optimal solution is detected based on the condition given in (76), the deterministic guided mechanism described in section 4.2 will be used. The hybrid algorithm terminates when a preset time bound is reached or the current best solution has not been improved after δ iterations. The hybrid algorithm is called *Reactive Iterative Path Coordination* (RIPC) algorithm. The pseudo code of RIPC using the local probabilistic model $P_{\eta,\beta}(r|r^{cur})$ is given in Figure 6-1. The scheme **pcUpdate**() updates the value of μ using the adaptive μ strategy described in section 3.2.2 with the penalty cost $\Sigma(r)$ given in (51), while the scheme **betaUpdate**() updates the value of β using the adaptive β strategy suggested in section 5.2.3. The scheme **deltaUpdate**() sets the value of Δ_β according to the condition (80) every π iterations.

Algorithm 6-1: RIPC($r^{init}, r^{best}, P_{\eta, \beta}(r|r^{cur})$)

- 1) $r^{[0]} \leftarrow r^{init}$, $h_j \leftarrow 1$ for $j=1,2,\dots,n$;
- 2) $i \leftarrow 1$, $\tau \leftarrow 1$, $\alpha \leftarrow 0$, $\eta \leftarrow 1$, $\Delta_\beta \leftarrow \varepsilon_\Delta$;
- 3) Repeat the following procedure until some stopping criteria are satisfied:
 - a) $\mu \leftarrow pcUpdate()$;
 - b) $\Delta_\beta \leftarrow deltaUpdate()$;
 - c) At the iteration τ , repeat the following procedure until a new solution r' is successfully generated:
 - i. $\beta \leftarrow betaUpdate()$;
 - ii. $r^{[\tau]} \leftarrow ProbabilisticPathPlanner(N_i(r^{[\tau-1]}), P_{\eta, \beta}(r|r^{[\tau-1]}))$;
 - d) If $r^{[\tau]} = r^{[\tau-1]}$, then $\alpha \leftarrow \alpha + 1$, or else $\alpha \leftarrow 0$;
 - e) If $\alpha \geq n$, then
 - i. $j_{max} \leftarrow maxIndex_{1 \leq j \leq n} \{U(r^*, j)\}$;
 - ii. $h_{j_{max}} \leftarrow h_{j_{max}} + 1$;
 - iii. $\alpha \leftarrow 0$;
 - f) If $r^{[\tau]}$ is feasible:
 - i. If $F(r^{best}) > F(r^{[\tau]})$ or $r^{best} = \emptyset$, then $r^{best} \leftarrow r^{[\tau]}$;
 - ii. $\eta \leftarrow \eta_{min}$;
 - g) If $r^{[\tau]}$ is not feasible, $\eta \leftarrow 1$;
 - h) $i \leftarrow i + 1$ if $i < n$, otherwise $i \leftarrow 1$;
 - i) $\tau \leftarrow \tau + 1$;
- 4) Return $r^{[\tau-1]}$;

Figure 6-1. Pseudocode for the Reactive Iterative Path Coordination (RIPC) algorithm for MAPP.

6.2 Solving MAPP Optimization Problems using RIPC

6.2.1 Reactive Iterative Makespan Minimization for Makespan Minimized MAPP

Figure 6-2. Overview description of the Reactive Iterative Makespan Minimization (RIMM) algorithm for solving makespan minimized MAPP.

The iterative algorithm for solving makespan minimized MAPP using RIPC is shown in Figure 6-2. Beginning with an initial solution $r^{(0)}$, which is given by r^{init} , the optimal makespan minimized MAPP solution is approximated by solving a sequence of minimization problem in (36) with the control parameter c gradually increased from 0 based on an increment rate of ϵ_c . At each iteration k , the minimization problem in (36) is solved by $\mathbf{RIPC}(r^{(k-1)}, r^{best}, P_{\eta, \beta}(r|r^{cur}))$ with the local probabilistic model $P_{\eta, \beta}(r|r^{cur})$ satisfying **Condition 2** described in section 5.2.3 under the settings in (53) and (54). The iterative algorithm using RIPC for makespan minimized MAPP is called Reactive Iterative Makespan Minimization (RIMM). RIMM is made to terminate when the current best solution has not been improved after δ iterations within RIPC or a preset time bound is reached.

6.2.2 Reactive Iterative Cumulation Minimization for Total Cost Minimized MAPP

Figure 6-3. Overview description of the Reactive Iterative Cumulation Minimization (RICM) algorithm for solving total cost minimized MAPP.

The iterative algorithm for solving total cost minimized MAPP using RIPC is shown in Figure 6-3. Beginning with the initial solution $r^{(0)}$, which is given by r^{init} , the optimal solution of the total energy cost minimized MAPP problem is approximated by iteratively solving the minimization problem in (42). At each iteration k , the minimization problem in (42) is solved by $\mathbf{RIPC}(r^{(k-1)}, r^{best}, P_{\eta, \beta}(r|r^{cur}))$ with the probabilistic model $P_{\eta, \beta}(r|r^{cur})$ satisfying **Condition 2** in section 5.2.3 under the settings in (56) and (57). This iterative algorithm using RIPC for total cost minimized MAPP is called Reactive Iterative Cumulation Minimization (RICM). Like RIMM earlier, RICM is also made to terminate when the current best solution has not been improved after δ iterations within RIPC or a preset time bound is reached.

6.3 Experimental Evaluation

The performance of the RPC algorithm was first evaluated under different parameter settings, and then compared with the GIPC, PIPC and other existing MAPP algorithms in two different aspects. Firstly, these comparative MAPP algorithms were compared in grid maps to investigate their general performances when used for solving the makespan minimized and total cost minimized MAPP problems. Secondly, they were compared in solving the challenging MAPP problem scenarios highlighted in Chapter 1 to evaluate their ability of finding a feasible MAPP solution, which is the first and primary objective in this work.

6.3.1 Evaluation of RPC with Increasing Allowable Computational Time

A parameter setting strategy is proposed for the increment rate Δ_β of adaptive β strategy in the RPC algorithm to facilitate a switch of navigation strategy when searching in the infeasible region. To fine tune the parameter Δ_β , additional new parameters like the fixed value ε_Δ , the sample size π for the infeasible iterative solutions and the possibility value p were introduced to control the increment rate Δ_β for adaptive β in RPC. The fixed value of $\varepsilon_\Delta=0.2$ was observed to be a good choice for Δ_β when evaluating the performance of PIPC in section 5.4.1.1 and it is also adopted in RPC. However, the other two parameters, namely the sample size π and possibility value p , need further investigation. The following experiments in this section evaluate the performance of RPC under varying possibility value p and sample size π when used within the RIMM algorithm for solving the makespan minimized MAPP problem. The evaluations were done with the parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for RPC and $\varepsilon_c=1$ for control parameter c in RIMM.

6.3.1.1 Evaluation of RIPC with different possibility values p

Being a threshold value in (78) that decides whether RIPC is deemed to be trapped in an infeasible local optima basin, the possibility value p plays an important role in reliably detecting such infeasible locally optimal basins during the search process. Section 6.1.1 proposes two ways for setting the possibility value p , namely the fixed value and the heuristic value of p . The following experiments evaluate the performance of RIPC under a heuristic value of $p=0.05*\rho_p^5$ and two different fixed values of p , namely the lowest fixed value of $p=0.0$ and a high fixed value of $p=1.0$. The sample size π is set to n , the number of agents. To evaluate the reliability of the probabilistic detection mechanism in RIPC, a moderately and a highly crowded environment were used. The former is a 30x20 grid map with a fixed agent count of 200 and 20% static obstacle coverage (similar to the Figure 3-4) and the latter is a 4x4 grid map with a fixed agent count of 15 (see Figure 5-12).

Figure 6-4. Success rate of RIPC with different possibility values p against allowable computational time in a 30x20 grid map (moderately crowded environment).

Figure 6-5. Task completion time of RIMM with different possibility values p against allowable computational time in a 30x20 grid map (moderately crowded environment).

Figure 6-6. Success rate of RIMM with different possibility values p against allowable computational time in a 4x4 grid map (highly crowded environment).

Figure 6-7. Task completion time of RIMM with different possibility values p against allowable computational time in a 4x4 grid map (highly crowded environment).

When p is set to the lowest fixed value (ie. $p=0.0$), the probabilistic detection mechanism will always decide that RIPC gets trapped in an infeasible locally optimal basin when searching in the infeasible solution region. In this case, RIPC will switch to the best improvement navigation strategy all the time before entering a feasible solution region. Since the best improvement navigation strategy in GIPC is more computationally costly than the probabilistic improvement navigation strategy in PIPC in finding a feasible MAPP solution (see experimental results in section 5.4.2), as expected, it was observed that in the 30x20 grid map, $p=0.0$ resulted in a slower increase of success rate with increasing allowable computational time compared with $p=1.0$ (see Figure 6-4). On the contrary, setting p to a high value could cause the probabilistic detection mechanism in RIPC to fail in its detection of being trapped in infeasible local optimal basins and therefore RIPC sticks to the probabilistic improvement navigation strategy while the guided mechanism described in section 4.2 is never employed. Under such circumstance, RIPC behaves like PIPC. In the highly crowded environment (the 4x4 grid map), RIPC with a high value of $p=1.0$ fails to

find a feasible MAPP solution. Thus, setting either a high fixed value or a low fixed value to p is not appropriate.

In contrast, the heuristic value of $p=0.05*\rho_p^5$ takes into account the information associated with the agent count and the size of traversable space. As the environment gets crowded, the value of p is set lower according to the formulation given by $p=0.05*\rho_p^5$. In so doing, RIPC can be made adaptive to different environments. It was observed that the proposed heuristic value of $p=0.05*\rho_p^5$ is able to exhibit fast increase in success rate and comparable task completion time performance as $p=1.0$ in a moderately crowded environment. While at the same time, exhibit comparable success rate and task completion time as $p=0.0$ in a highly crowded environment.

6.3.1.2 Evaluation of RIPC with different sample size π

Since the sample size π determines the number of sampled iterative solutions used for estimating the chance of being trapped in infeasible local optima and the frequency that the value of Δ_β is updated, the following experiments evaluate how different values of π influence the performance of RIPC. As a solution can only be claimed to be a locally optimal solution when it is the best solution within its n local neighborhoods, the smallest value of π has to be n . Thus, three different value of π were evaluated, namely the lowest value of $\pi=n$, an intermediate value of $\pi=10*n$ and a high value of $\pi=100*n$ (n is the number of agents). Based on the results in the previous section, the possibility value of p was obtained using the heuristic value of $p=0.05*\rho_p^5$. The evaluations were only done in the highly crowded environment (4x4 grid map with a fixed agent count of 15) since the switch of navigation strategy is only likely to happen in a highly crowded environment according to the probabilistic detection mechanism with the heuristic value of $p=0.05*\rho_p^5$.

Figure 6-8. Success rate of RIMM with different sample sizes π against allowable computational time in a 4x4 grid map.

Figure 6-9. Task completion time of RIMM with different sample sizes π against allowable computational time in a 4x4 grid map.

When π is set to a large value, the value of Δ_β is updated infrequently. This results in an unresponsive switch between the best improvement and probabilistic improvement navigation strategies. As such, a higher value of π was observed to produce poorer success rate and task completion time performances in the highly crowded environment (see Figures 6-8 and 6-9).

In summary, the choice of the possibility value p and the sample size π has a significant impact on the performance of RIPC in intelligently and reactively combining the techniques of PIPC and GIPC in different environments. Based on the observed results, the heuristic value of $p=0.05*\rho_p^5$ and $\pi=n$ are good parameter settings that will allow RIPC to quickly find superior MAPP solutions.

6.3.2 Comparative Evaluation of RIPC with GIPC and PIPC

This set of experiments compared the performance of RIPC with GIPC, PIPC under varying agent count in two different sized grid maps. The first one is the 4x4 grid map (see Figure 5-12) with agent count ranging from 2 to 15. The varying environments from a sparsely crowded to an extremely crowded scenario in the 4x4 grid map allow the overall performance of the comparative MAPP algorithms to be compared while the small map size of 4x4 facilitates the experimental results to be obtained within a reasonable computational time. The second is a much larger size grid map of 30x20 (see Figure 3-4) with a static obstacle density of 20% and agent count varying from 20 to 200. The larger sized 30x20 grid map has a larger size of traversable space and therefore allows for a larger number of agents. As such, the larger environment allows the simulation of a larger size of solution space and facilitates the investigation of the performance of these comparative MAPP algorithms as the size of solution space increases.

The three proposed local search algorithms, namely GIPC, PIPC and RIPC were only compared when used for solving the makespan minimize MAPP problem. The comparisons were done under the same time bound of 5 minutes with the common parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for GIPC, PIPC and RIPC, the increment rate $\varepsilon_c=1$ for control parameter c in GIMM, PIMM and RIMM. In addition, two iteration

bounds of $\delta=n$ and $\delta=15*n$ (n is the number of agents) on the improvement of the solution quality is imposed on the stochastic algorithms PIMM and RIMM, the adaptive η and β strategies were both used in PIPC and RIPC while the specific parameter settings of $\pi=n$, $p=0.05*\rho_p^5$ were only used for RIPC.

Figure 6-10. Comparative success rates of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.

Figure 6-11. Comparative task completion times of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.

Figure 6-12. Comparative computational times of GIMM, PIMM and RIMM under varying agent count in a 4x4 grid map.

Figure 6-13. Comparative success rates of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.

Figure 6-14. Comparative task completion times of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.

Figure 6-15. Comparative computational times of GIMM, PIMM and RIMM under varying agent count in a 30x20 grid map.

PIMM's poorer success rate than GIMM in a highly crowded environment (e.g. agent count > 12 in the 4x4 grid map – see Figure 6-10) was earlier observed in the experimental results of section 5.4.3.1. In these experiments, it is GIMM's turn to suffer poorer success rate performance when the traversable space and agent count increases significantly in the larger 30x20 grid map. The reason for this degradation in performance is due to GIMM's use of the GIPC local search algorithm. GIPC performs an exact search in the local neighborhood to

find an improving MAPP solution in contrast to PIPC's fast probabilistic generation on an improving neighboring solution. As such, GIPC's computational cost increases relatively quickly with increasing size of traversable space and agent count. As a result, GIMM's success rate dropped as the agent count increases to 200 (see Figure 6-13) because of a premature termination of the search when the preset time bound of 5 minutes is reached (see Figure 6-15). In short, GIMM works better in a small highly crowded environment while PIMM works better in a large moderately crowded one. The proposed hybrid algorithm RIMM ($\delta=n$) was observed to produce similar performance with PIMM ($\delta=n$) when the environment is not too crowded (map density < 50%) and performed like GIMM as the environment gets highly crowded. As a result, RIMM ($\delta=n$) exhibited the best overall performance compared with GIMM and PIMM ($\delta=n$). The reason for this is due to the use of the hybrid RIPC local search algorithm in RIMM. As the environment gets crowded, the probability that RIPC gets trapped in infeasible locally optimal basins increases. Under the proposed probabilistic detection mechanism and the switching scheme, RIPC begins with the probabilistic improvement navigation strategy and will continue to stay with this navigation strategy in a sparsely crowded environment. However, as the environment gets more crowded, it is more likely to switch between the probabilistic and best improvement navigation strategies. In this case, RIPC is able to switch between the best characteristics of GIPC and PIPC and therefore extract the best overall performance of the proposed local search algorithms.

It was observed in Figure 6-11, that the task completion time of RIMM ($\delta=n$) is consistently higher than that of GIMM. The task completion time performance of RIMM can be further improved if more iterations are permitted, like is seen in the plot of RIMM ($\delta=15*n$) but

Unfortunately, this improvement comes at the expense of higher computational time (see Figure 6-12).

6.3.3 Comparative Study of RIPC-based Algorithms and Other Existing MAPP Algorithms

The next set of experiments compared the performance of the two RIPC-based algorithms with the existing MAPP algorithms under varying agent count in the 4x4 and 30x20 grid maps used in section 6.3.2. The comparisons between these MAPP algorithms were done under the same time bound of 5 minutes with the parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$), adaptive η and β , sample size $\pi=n$, possibility value $p=0.05*\rho_p^5$ for RIPC, the increment rate $\varepsilon_c=1$ for control parameter c in RIMM, the cost ratio $\rho_c=0.1$ for RICM. In addition, an iteration bound of $\delta=n$ (n is the number of agents) on the improvement of the solution quality is imposed on RIMM and RICM.

6.3.3.1 Other MAPP algorithms used in comparative study

Since two different MAPP optimization problems were addressed in this thesis, the proposed local search algorithms that were specifically designed to solve the makespan minimized and total cost minimized MAPP problems were separately compared with appropriate existing MAPP algorithms that were designed to address the same optimization objective.

For makespan minimized MAPP, two comparative MAPP algorithms were chosen. The first one is the Yu and LaValle's Integer Linear Programming (ILP) based algorithm [44], which is a complete algorithm and is specifically designed to solve the makespan-minimized MAPP problem. ILP is able to find an optimal makespan MAPP solution if enough computational time is allowed. Comparison with ILP therefore allows the quality of the solution obtained by the proposed local search algorithms to be benchmarked to ground truth results (if sufficient

computational time is provided). The second comparative algorithm is the well-known Windowed Hierarchical Cooperative A* (WHCA*) algorithm proposed by Silver [26], which moves the agents to their respective destination concurrently but makes a tradeoff between the solution quality achieved and the computational complexity of the algorithm. By sacrificing optimality, WHCA* is supposed to be computationally more efficient in finding a reasonably good MAPP solution. Comparison with WHCA* allows the computational efficiency of the proposed algorithms for makespan minimized MAPP to be compared alongside the quality of their respective task completion times. Comparison was done with WHCA* using different window steps, which was selected appropriately based on the size of the grid map under evaluation.

For total cost minimized MAPP, two comparative MAPP algorithms were also chosen. The first one is a recently proposed Optimal Anytime (OA) algorithm proposed by Standley and Korf [47]. OA is designed to find an optimal MAPP solution with respect to a cumulative cost function. Like the proposed iterative algorithms in this work, OA is also an anytime algorithm that can improve its solution quality if more computational time is permitted. As such, OA is a good benchmark for comparing the relative superiority of the energy efficient MAPP solution that can be obtained for a given allowable computational time. The other comparative algorithm is a computational efficient algorithm proposed by Luna and Bekris [39] called Push and Swap (P&S). P&S has excellent success rate and computational time performance as it moves one agent at a time and computes MAPP solutions in polynomial time. P&S therefore serves as a good benchmark for comparing the relative cumulative cost performance between a fast polynomial time algorithm and the proposed anytime algorithms that are specifically designed for total cost minimized MAPP.

All benchmark algorithms used were re-implemented based on published works except ILP [49], which was modified from available source code obtained from [100].

6.3.3.2 Comparative evaluation of RIMM for makespan minimized MAPP

The following experiments compare the performance of RIMM with the ILP and Windowed Hierarchical Cooperative A* with 16 window steps (WHCA*(16)) and 128 window steps (WHCA*(128)) algorithms in solving makespan minimized MAPP on a 4x4 and 30x20 grid map separately.

Figure 6-16. Comparative success rate of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-17. Comparative task completion time of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-18. Comparative computational time of the makespan minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-19. Comparative success rate of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Figure 6-20. Comparative task completion time of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Figure 6-21. Comparative computational time of the makespan minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Being an optimal MAPP algorithm for makespan minimized MAPP, ILP suffers from rapidly increasing search space with the agent count and the size of traversable space. This can be seen in Figure 6-18 and Figure 6-21, where ILP is observed to have the highest computational cost compared with other makespan minimized MAPP algorithms. Due to the intentional choice of the low agent count and a small map size of only 4x4, ILP was observed to have stayed within the imposed timeout of 5 minutes and was therefore able to achieve 100% success rate and also exhibit the best task completion performance in the 4x4 grid map. Unlike the 4x4 grid map, the size of the solution space increases more significantly in the larger 30x20 grid map as it has a larger size of traversable space and allows for a higher agent count. As such, despite the high timeout of 5 minutes, ILP's success rate drops quickly as the agent count increases in the 30x20 grid map (see Figure 6-19). Since ILP is an algorithm that is specifically designed to find the optimal MAPP makespan solution, its task completion time performance can effectively be considered the “gold standard” performance against which all other algorithms can be benchmarked.

By sacrificing the optimality and completeness of the algorithm, WHCA* was indeed more computationally efficient than ILP in finding a feasible MAPP solution (see Figures 6-18 and 6-21). However, its success rate dropped quickly as the environment gets moderately crowded (agent count >4 but <12 in the 4x4 grid map and agent count >100 in the 30x20 grid map), and its task completion time performance was observed to be the worst compared to all other comparative MAPP algorithms (see Figures 6-17 and 6-20). The larger window step of WHCA* may help the coordination of paths among agents and therefore increase its success rate, but the improvement can only be made if the window step is smaller than the length of a viable path. This is because WHCA* will perform like HCA* [26] as the window step is sufficiently long for a complete path planning. As a result, in the 4x4 grid map, the sufficiently long window step of WHCA*(128) was observed to have a lower success rate compared with WHCA*(16) (see Figure 6-16). In contrast, in the 30x20 grid map, 128 window steps is not sufficiently long for a complete path planning. In this case, WHCA*(128) could make an improvement on its success rate performance compared with WHCA*(16) (see Figure 6-19). However, a larger window step led to a higher computational cost (see Figures 6-18 and 6-21) and a worse task completion time performance (see Figures 6-17 and 6-20).

As an algorithm that are specifically designed for makespan minimized MAPP, RIMM ($\delta=n$) was observed to be more computationally fast in finding a good makespan solution compared with ILP. Though task completion time performance of RIMM ($\delta=n$) is still inferior to that of ILP, it can be further improved by using a higher iteration bound ($\delta>n$), which translates to a higher computational cost. Being a decoupled approach like WHCA*, RPC also performs a search in a reduced search space each time. Unlike the constructive nature [111] of WHCA*, RPC allows the solution quality to be refined iteratively. It was observed that

RIMM ($\delta=n$) was able to achieve as good a task completion time as WHCA*(16) while outperforming WHCA* in the other two performance criteria, namely the success rate and computational time. As a result, as an anytime algorithm for makespan minimized MAPP, RIMM can strike a good tradeoff between the solution quality and the computational time incurred. In addition, RIMM was also observed to be able to achieve this in both small highly crowded and large moderately crowded environments.

6.3.3.3 Comparative evaluation of RIMM for total cost minimized MAPP

The following experiments compare the performance of RIMM with the GICM, PICM, OA and P&S algorithms in solving total cost minimized MAPP on a 30x20 grid map and a 4x4 grid map separately.

Figure 6-22. Comparative success rate of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-23. Comparative cumulative cost of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-24. Comparative computational time of the total cost minimized MAPP algorithms under varying agent count in a 4x4 grid map.

Figure 6-25. Comparative success rate of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Figure 6-26. Comparative cumulative cost of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Figure 6-27. Comparative computational time of the total cost minimized MAPP algorithms under varying agent count in a 30x20 grid map.

Being an optimal algorithm for total cost minimized MAPP, OA is able to find an energy efficient MAPP solution if enough computational time is permitted. Like OA, the proposed RISM algorithm was also designed specifically to solve the total cost minimized MAPP problem. It was observed to be able to match the cumulative cost performance of the optimal OA algorithm in the smaller 4x4 grid map (See Figure 6-23). However, the state search space of OA increases rapidly with increasing agent count and depth of the goal state. And the depth of the goal state increases rapidly as the map density and the size of traversable space increases. Within the time bound of 5 minutes, OA's success rate drops quickly as the agent count increases, even in the smaller 4x4 grid map (see Figure 6-22). The computational time performance is a major advantage of RISM over optimal algorithms such as OA and this allow RISM to be employed in user-interactive applications such as those highlighted in Figure 1-4.

Being a polynomial time algorithm, P&S exhibits excellent success rate and computational time performances when the environment is not extremely crowded. However, as the environment gets extremely crowded (i.e. agent count=15 in the 4x4 grid map), the success

rate of P&S begins to fall. Given that the P&S algorithm requires at least two empty vertices in the map, it was not expected to find viable paths for all n agents when $n = 15$, as observed in Figure 6-22. In contrast, RICM is able to maintain much better success rate performance in this extremely crowded scenario. In addition, since the sequential algorithm P&S plans the path for one agent at a time, it does not produce cumulative cost performances that can match those of RICM. (see Figure 6-23 and Figure 6-26). In fact, the disadvantage of sequential algorithms like P&S is even more serious if the MAPP application requires makespan minimized solutions. RICM's ability to move all n agents simultaneously will complete the task in a significantly shorter time than P&S, as is observed in the comparative task completion time performance in Figure 6-28.

Figure 6-28. Comparative task completion times of the P&S and RICM algorithms under varying agent count in a 30x20 grid map.

6.3.4 Comparative Evaluation of RIPC in Solving Challenging MAPP Scenarios

This section presents an interesting set of experiments evaluating the ability of the various MAPP algorithms in resolving some specific MAPP problem scenarios such as those highlighted earlier in Chapter 1, namely the MAPP problems in the Loop (see Figure 6-29(a)), Tunnel (see Figure 6-29(b)) and Tree (see Figure 6-29(c)) scenarios. These are challenging scenarios because finding viable paths requires agents to have the notions of priority assignment, the ability to move to “waiting zones” and temporarily suffer non-improving solutions in order to allow other agents to achieve their objectives.

Figure 6-29. The three challenging MAPP problem evaluated, namely (a) Loop, (b) Tunnel and (c) Tree scenarios. The top row shows the simulation environment used for the experiment and the bottom row shows the equivalent graphical model of each of the respective MAPP scenarios.

The comparisons between these MAPP algorithms were done under the same time bound of 5 minutes with the specific parameter settings of $\pi=n$, $p=0.05*\rho_p^5$ for RIPC and the common parameter settings of adaptive μ ($a_A=0.1$, $\omega=5*n$, $\lambda_0=0.05$ and $\mu_0=\mu_{min}$) for GIPC,

PIPC and RPC while the adaptive η and β strategies for PIPC and RPC. Due to a low map size, the 8 window steps and 16 window steps were used for WHCA*.

Table 6-1. Comparative results of the MAPP algorithms in solving challenging MAPP scenarios. For successful scenarios, their computational times (rounded to the nearest millisecond) are given in brackets.

MAPP Scenarios	MAPP Algorithms							
	ILP	OA	PIPC	RIPC	PS	GIPC	WHCA* (8)	WHCA* (16)
Loop	Success (56 ms)	Success (1 ms)	Success (<1 ms)	Success (<1 ms)	Failure	Success (1 ms)	Failure	Success (2 ms)
Tunnel	Success (25 ms)	Success (<1 ms)	Success (1 ms)	Success (<1 ms)	Success (1 ms)	Failure	Success (1 ms)	Success (1 ms)
Tree	Success (56 ms)	Success (1 ms)	Success (1 ms)	Success (1 ms)	Failure	Failure	Failure	Failure

It is not surprising that optimal algorithms like ILP and OA were able to find feasible MAPP solutions in all three scenarios. Given that these scenarios have a low agent count of 2 or 3 and a very small map size, these optimal algorithms that do not scale well computationally had no issue in finding the feasible solution with the allocated the time bound of 5 minutes. As ILP is an optimal algorithm that is designed to find a feasible and optimal solution while OA is an anytime optimal algorithm that is able to refine its solution quality with increasing computational cost, in the case where only a feasible solution is required, OA can be more computationally efficient than ILP (see Table 6-1).

As a polynomial time algorithm for MAPP, P&S was claimed to be complete given the assumption that two empty vertices are provided in Luna and Bekris [39]. However, the claim has been proven incorrect by Wilde in [43] and the experiments in this section lend further evidence of this. The MAPP problems in the Loop and Tree scenarios are two

examples that P&S failed even if two or more empty vertices are provided. For the loop scenario shown in Figure 6-29(a), the failure of P&S is due to the shortest path used in the push operation. The agent A1 can directly go ahead and reaches its destination along its shortest path by pushing A2 to the right. When the agent A2 is on, A1 blocked the shortest path of A2. In this case, a swap operation is required for A2 to reach its destination. However, since no vertex with three degrees exists in the Loop scenario, the swap operation will fail. As for the MAPP problem in the tree scenario shown in Figure 6-29(c), the failure of P&S is due to the planning order of the agents. The agent A1 has already stayed in its destination. Then for the agents A2 and A3, swap operations are required. Though a vertex with three degrees exists in the junction, it is impossible for clear operation to leave two empty unoccupied vertices around the junction. As such, the swap operation will again fail. This same weakness in P&S was observed earlier in the experimental results in Figure 6-22, where the success rate in P&S started to drop when the agent count is 14 in the 4x4 grid map.

As another computationally fast MAPP algorithm, WHCA* compromises its completeness for computational speed. In the Loop scenario shown in Figure 6-29(a), WHCA*(8) failed since the short 8 window steps is not large enough for the agent A2 to choose an alternative longer collision-free path. In this case, the two agents A1 and A2 takes turn pushing each other away from their destinations and therefore WHCA*(8) fails to find a stable MAPP solution. In contrast, WHCA*(16) succeeded as the 16 window steps is large enough for the agent A2 to find the alternative longer collision-free path from its initial position to its destination. In the challenging Tree scenario shown in Figure 6-29(c), WHCA*(8) and WHCA*(16) both failed due to the drawback of prioritized planning approach that the high priority agent does not consider the path planning process of low priority agents. In this scenario, the agent A2 ignored the positions of other agents and could reach its destination in

the period when it is dynamically assigned the highest priority. In this case, the agent A3 failed to find any viable path as the agent A3 has no place to move so as to avoid the collision with the agent A2.

It was observed that GIPC failed in solving the Tunnel (see Figure 6-29(b)) and Tree (see Figure 6-29(c)) scenarios. This is because GIPC suffers from indeterminable steady state convergence characteristics. As shown in Figure 6-29(b), GIPC failed due to competing for the high priority between the two agents A1 and A2 at the middle node. This lead to a cyclical increase of the solution feature penalty associated with these two agents and therefore does not produce a stable MAPP solution in GIPC. Similarly in the Tree scenario, GIPC also has the same problem with the agents A2 and A3. This is the same reason why the earlier results in Figure 6-10 showed that the GIPC-based algorithm was unable to achieve 100% success rate when the agent count increases to 15 in the 4x4 grid map.

The stochastic nature of selecting neighboring solutions allows PIPC to explore any solution with a probability and this therefore facilitates the extensive search within the solution space. As such, given sufficient computational time, PIPC is likely to find a feasible solution if it exists. PIPC was observed to be able to find solutions for all the three scenarios in Figure 6-29 with a low computational time of 1 millisecond or less. Like PIPC, RIPC was also observed to be able to solve all three scenarios within 1 millisecond. Being a hybrid local search algorithm, RIPC allows the switch between the probabilistic and deterministic navigation strategies. The use of the probabilistic solution generation mechanism in RIPC allows perturbation to the solution search and is able to avoid the monotonic convergence behavior of deterministic algorithms like GIPC.

6.4 Summary

This chapter first proposed a hybrid local search algorithm called Reactive Iterative Path Coordination (RIPC), which combines GIPC's guided technique and PIPC's probabilistic solution generation mechanism for MAPP. Similar with GIPC-based and PIPC-based iterative algorithms, two RIPC-based iterative algorithms namely Reactive Iterative Makespan Minimization (RIMM) and Reactive Iterative Cumulation Minimization (RICM) were proposed by replacing IPC with RIPC within the iterative frameworks in Figures 3-2 and 3-3 individually. The RIPC-based algorithms were evaluated under different parameter settings and then compared with the GIPC-based, PIPC-based and other existing MAPP algorithms under different environment characteristics. It was observed that the RIPC-based algorithms were able to achieve the best overall performance of all three local search based algorithms proposed in this work due to its used hybrid local search RIPC. With the help of a probabilistic detection mechanism that detects the situation of being trapped in infeasible locally optimal basins and a switching scheme that is able to perform a switch between the probabilistic and best improvement navigation strategies, RIPC is able to allow these two combined techniques to work effectively in different environments. As such, RIPC is able to benefits from the PIPC's computational efficiency in finding an improving neighboring solution and GIPC's superior ability in escaping from infeasible locally optimal basins.

7 CONCLUSION AND FUTURE WORK

The work described in thesis was an attempt to address the multi-agent path planning needs of an interactive educational systems comprising of multiple cube-shaped mobile robots controlled by a central planner. This rather unique MAPP application scenario needed computationally responsive algorithms that can successfully produce good quality makespan minimized and total cost minimized MAPP solutions under situations where the working environment may be relatively large and is populated by many mobile and static agents. In addressing this challenge, several novel contributions and observations were made regarding how robust and computationally-efficient MAPP optimization algorithms can be designed.

7.1 Summary of Contributions

The first contribution is the incorporation of a local search technique into an iterative framework for addressing makespan minimized and total cost minimized MAPP problems. The proposed local search algorithm called Iterative Path Coordination (IPC) avoids the problem of rapidly increasing computational cost with increasing agent count as it iteratively improves the solution quality by planning the path for one agent at a time. In addition, its memory requirement is also not increasing rapidly with increasing agent count since it proceeds each search step by only taking into account the current solution.

The second major contribution is the enhancement of the iterative framework with a guided local search technique that was able to address IPC's shortcoming of not being able to find feasible solutions in a moderately crowded environment. The proposed Guided Iterative Path Coordination (GIPC) algorithm is able to use guided techniques to steer the search away from infeasible locally optimal basins. To apply guided local search to the MAPP problem, a novel guided cost function and a problem-specific guided mechanism was proposed. The

CHAPTER 7. CONCLUSION AND FUTURE WORK

effectiveness of employing guided local search was evident from the superior success rate performance improvement observed over the IPC-based algorithms. Unfortunately, the issue of computational efficiency of the GIPC-based algorithms was still a concern, especially in large-sized environments that are populated with many mobile agents. It was observed that the exact search for the best improving solution in the local neighborhood was a significant contributor to the computation cost of the GIPC-based algorithms. The notion of adopting a stochastic approach to solve the MAPP optimization problem was then explored.

The third major contribution in this thesis is the attempt to address the shortcoming of IPC being trapped in infeasible locally optimal basins by using a stochastic technique that allows non-improving solutions to be accepted with some probability. This led to the proposal of the Probabilistic Iterative Path Coordination (PIPC) algorithm. In order to model the MAPP problem in probability space, a novel local probabilistic model that generates neighboring MAPP solution probabilistically was proposed. By generating each neighboring solution probabilistically according to a local probabilistic model that is non-increasing with the target cost function, the PIPC algorithm can be made to converge to a locally optimal solution with the highest probability. Compared to GIPC, PIPC performs a fast probabilistic search in each local neighborhood and is therefore more computational efficient than GIPC in finding a neighboring MAPP solution. Unfortunately, this very probabilistic approach of accepting non-improving solutions makes PIPC inefficient in escaping from infeasible locally optimal basins. As a result, the computational efficiency it was designed to exhibit quickly degrades in highly crowded environments where the likelihood of the search going into infeasible locally optimal basins is high.

Given the fact that both GIPC and PIPC have their respective strengths and weaknesses in different operating environments, the final challenge was to see if a hybrid algorithm could

be devised. This algorithm must be able to combine GIPC's guided cost function and guided mechanism with PIPC's probabilistic solution generation mechanism. To allow the hybrid algorithm to work effectively, a novel probabilistic detection mechanism and a problem-specific switching scheme were proposed to enable the solution search process to reactively switch between the best and probabilistic improvement navigation strategies when the conditions favorable to the appropriate strategy is encountered. The proposed Reactive Iterative Path Coordination (RIPC) based algorithms were observed to exhibit the PIPC-based algorithms' superior performance in a sparsely to moderately crowded environment while behaving more like the GIPC-based algorithms in a highly crowded environment where GIPC works better. Thus, RIPC is able to benefit from PIPC's computational efficiency in finding an improving neighboring MAPP solution and GIPC's superior ability in escaping from infeasible locally optimal basins.

One pertinent observation of the proposed local search algorithms was the introduction of numerous parametric variables or settings. This is an unavoidable outcome of the formulation used to incorporate penalty coefficient in the constrained optimization problem or the relaxation and minimization parameters needed to define the success probabilities of free and collision nodes in the local probabilistic model of PIPC. Experimental observations also suggest that there is no easy way to fix the values of these parameters as their settings have varying and conflicting effects on the different performance criteria. For example, setting the penalty coefficient μ to a high value may yield superior success rate performance but at the detriment of the task completion time performance. For this reason, another noteworthy contribution of this work is the formulation of numerous adaptive parameter setting strategies (e.g. adaptive μ , η and β) that was experimentally shown to be superior to simple fixed parameter settings. These adaptive strategies were designed to automatically adjust the values

of the performance sensitive parameters based on current and historical information obtained during the search process. In essence, the adaptive tuning of these parameters allowed the incorporation of heuristics that can dynamically switch between an extensive or intensive search strategies based on the current state of the local search. This reactive adaption significantly enhances the efficiency of the iterative process of searching for a solution with improving quality.

Figure 7-1. Overview description of the iterative algorithm for solving makespan minimized MAPP.

Looking back to the iterative frameworks for solving makespan minimized MAPP in Figures 3-2, 4-2, 5-3, 6-2, the details of each common iterative step in the flow chart have been described in section 3.3.1, the only difference between them is their employed local search algorithms as shown in Figure 7-1. Similarly with the iterative frameworks for solving total cost minimized MAPP in Figures 3-3, 4-3, 5-4, 6-3, by using a different local search algorithm proposed in this work, the whole iterative algorithm is called a different name.

CHAPTER 7. CONCLUSION AND FUTURE WORK

One of the issues associated with the proposed iterative algorithms for makespan minimized and total cost minimized MAPP is their scalability. Assume m is the size of the traversable space in the four connected grid map and n is the number of agents. Since the computational time performance of the iterative algorithms mainly depends on the number of iterations τ_{max} required by its employed local search algorithm and the computational cost within each iteration of the local search algorithm, its worst case computational cost could be $O(\tau_{max} * m * T_{max})$, where T_{max} is the maximal time step that the local search algorithm has explored. The parameter τ_{max} is quite related to the ratio of the agent count n to the size of traversable space m and it was observed that τ_{max} has a near linear relationship with the ratio in a sparsely and moderately crowded environment but it quickly increases with the increasing ratio in a highly crowded environment. The parameter T_{max} will increase fairly with the increasing size of traversable space m but quickly with the increasing ratio of the agent count to the size of traversable space. As a result, the proposed iterative algorithms may have a quickly increasing computational cost as the environment gets crowded but has a fairly increasing computational cost with the increasing size of traversable space. Thus, compared with the agent count, the proposed iterative algorithms are more scalable to the map size.

Another issue is the fact that unsolvable MAPP problems cannot be detected by the proposed iterative algorithms. This is why a preset time bound is imposed to terminate the algorithm. An alternative way of tackling this issue is to employ a complete polynomial time algorithm to test for the MAPP feasibility and then provide the initial feasible solution for the iterative algorithms.

7.2 Future Work

Though the proposed three variants of the IPC algorithm are able to find a sufficiently good MAPP solution quickly, there is still future work that can be explored to improve their performance.

Firstly, GIPC was specifically designed to address the problem of being trapped in infeasible locally optimal basins. However, once a feasible local optimum has been reached, no further improving solution will be sought. This prevents GIPC from finding better MAPP solutions, even if more computational time is permitted. A possible future investigation is the design of a new guided technique that incorporates some optimality conditions so as to guide the search out of feasible locally optimal basins. To realize this, some challenging issues need to be addressed, such as the choice of the optimality condition, the definition of its associated solution feature and the subsequent design of an efficient guided cost function.

With the local probabilistic model that is specifically designed for MAPP, the PIPC algorithm is governed by many performance-sensitive parameters such as μ , η and β . Though adaptive parameter setting strategies have been suggested to automatically adjust the values of these parameters individually, several issues remain to be solved. Firstly, the possibility of tuning multiple parameters simultaneously has not been investigated in this work. This is desirable due to the correlation among the inherent parameters μ , η and β in PIPC. Since the optimal setting of each of these parameters may conflict with each other, pareto-based multi-objective approach could be a promising way to find a good tradeoff value setting. Secondly, the design of these strategies has inadvertently introduced more new parameters such as the increment rate Δ_β and some threshold values of such as ε_1 in (69) and ε_2 in (70). Though suitable fixed values have been suggested for these parameters through empirical studies,

CHAPTER 7. CONCLUSION AND FUTURE WORK

there exist further opportunities to further improve the overall performance of the PIPC algorithm by designing more informed and adaptive parameter setting strategies for these parameters too. The challenge is to figure out what are the search state and environmental information needed to design effective heuristics for determining the values of these parameters. Empirical observations seem to suggest that the design of these possible informed parameter setting strategies may require taking into account information such as the map density and the size of traversable space. In contrast with the problem specific heuristic idea based on empirical observations, a general methodology can be employing learning techniques such as reinforcement learning [121] to automatically tune the parameters during the search process.

Another possible area of improvement is in the hybrid RIPC algorithm. It was observed that GIPC has a faster convergence rate towards an optimal solution compared with PIPC. This particular strength of GIPC has not been fully exploited by the current design of the RIPC algorithm. A possible future direction is to design a more effective switch between the navigation strategies such that RIPC can also benefit from GIPC's fast convergence rate towards an optimal solution while maintaining PIPC's computational efficiency in searching for an improving neighboring solution.

The proposed local search algorithms have been demonstrated their strengths when used within the iterative algorithms for solving makespan minimized and total cost minimized MAPP optimization problems. However, these two optimization criteria have been addressed separately. Given that the MAPP application scenario motivating this work requires the planning of multi-agent paths that can be traversed quickly and also in an energy-efficient manner, these two problems should ideally be tackled as a single problem formulation. An important future direction is to investigate if both makespan and total cost MAPP can be

simultaneously minimized as a bi-objective optimization problem [112]. The earlier examples in Figures 1-5 and 1-6 have shown a single solution that is both makespan minimized and total cost minimized does not exist. As a result, unlike single-objective optimization problem like makespan minimized MAPP or total cost minimized MAPP, the bi-objective optimization problem named makespan and total cost minimized MAPP may have a set of non-dominated MAPP solutions that are not better than each other. These non-dominated solutions are also called Pareto optimal solutions [114]. In general, a bi-objective optimization problem is desired to be solved by finding all or a set of representative Pareto optimal solutions. Mathematical programming-based a posteriori methods like [115], [116], [117] and evolutionary algorithms like [118], [119] can be used to solve bi-objective optimization problems. However, for the makespan and total cost minimized MAPP problem, evolutionary algorithms suffers from the memory issue as it requires a large population of MAPP solutions that is sufficient to generate an even set of Pareto optimal solutions. In contrast, mathematical programming-based a posteriori methods are sensitive to the non-smoothness of the makespan objective function. What is more, they can not directly applied for solving discrete MAPP optimization problem. In this case, a single solution based stochastic multi-objective optimization algorithm like [120] could be a promising direction for solving the makespan and total cost minimized MAPP problem.

Apart from the previously mentioned directions for future work, some advanced future directions are worth investigating. Firstly, since the MAPP optimization problem in this work can be considered as a fully cooperative game, in which each agent or player has the shared utility function, so equilibrium seeking algorithms such as fictitious play [122] or other learning algorithms could possibly be used to solve the MAPP optimization problem. Secondly, the focus of this work belongs to the scope of classical planning, in which the

CHAPTER 7. CONCLUSION AND FUTURE WORK

actions are deterministic and the outcomes are predictable after any sequence of actions given a known initial state. However, in reality, the outcomes can be unpredictable due to the influence of nature or the existence of unpredictable moving obstacles in the environment. Considering issues of uncertainty when formulating solutions for the MAPP problem would make the research more relevant and useful to many more real-life application scenarios and is also worthy of future work.

PUBLISHED WORKS

- W. Wang and W. B. Goh, Spatio-Temporal A* Algorithms for Offline Multiple Mobile Robot Path Planning, in *Proceedings of 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2011)*, pp. 1091-1092, Taipei, Taiwan. 2-6 May 2011, 2011.
- W. Wang and W. B. Goh, Multi-robot Path Planning with the Spatio-Temporal A* Algorithm and Its Variants. *ARMS Autonomous Robots and Multi-robot Systems Workshop at AAMAS 2011 (ARMS2011)*, pp. 313-329, Taipei, Taiwan. 2-6 May 2011.
- W. Wang and W. B. Goh, Time Optimized Multi-Agent Path Planning using Guided Iterative Prioritized Planning, in *Proceedings of 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2013)*, Saint Paul, Minnesota, USA. 6-10 May 2013.
- W. Wang and W. B. Goh, A Meta-Heuristic Approach For Energy-Efficient Multi-Agent Path Planning, in *15th International Conference on Artificial Intelligence (ICAI2013)*, Las Vegas, USA, 22-25 July 2013.
- W. Wang and W. B. Goh, An Iterative Approach For Makespan-Minimized Multi-Agent Path Planning in Discrete Space, in *Journal of Autonomous Agents and Multi-Agent Systems (AAMAS)*. April, 2014. (DOI 10.1007/s10458-014-9259-z)

PUBLICATION UNDER REVIEW

- W. Wang and W. B. Goh, A Stochastic Algorithm For Makespan-Minimized Multi-Agent Path Planning in Discrete Space, in *Journal of Applied Soft Computing*.

REFERENCES

- [1] T. Lozano-Pérez, M. A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Communications of the ACM* 22 (10): 560–570, 1979.
- [2] C. O'Dunlaing, C. K. Yap, A “retraction” method for planning the motion of a disc, *Journal of Algorithms*, vol. 6, pages 104-111, 1982.
- [3] M. Weigl, B. Siemiatkowska, K. A. Sikorski and A. Borkowski, Grid-based mapping for autonomous mobile agent, *Agentics and Autonomous Systems*. Vol. 11, issue 1, pages 13-21, 1993.
- [4] S. M. LaValle, *Planning Algorithms*. Cambridge University Press. 2006.
- [5] 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. *IJRR*, 3(4): 76-88, 1984.
- [6] D. Ratner and M. Warmuth. The (n^2-1) -Puzzle and Related Relocation Problems. *J. Symbolic Computation*, 10, 111-137, 1990.
- [7] D. Kornhauser, G. Miller and P. Spirakis, Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science(FOCS 1984)*, pp. 241-250.
- [8] P. Surynek, An Optimization Variant of Multi-agent Path Planning is Intractable, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
- [9] M. Sipser, *Introduction to the Theory of Computation*, second edition, Cengage Learning, 2005.
- [10] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press and McGraw-Hill, Section 22.3: Depth-first search, pp. 540-549, 2001.

- [11] D. E. Knuth, The Art of Computer Programming, Vol. 1, 3rd ed, Boston: Addison-Wesley, 1997.
- [12] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* 1: 269–271, 1959.
- [13] N. J. Nilsson, Principles of Artificial Intelligence, Springer, 1982.
- [14] J. Kleinberg and É. Tardos, Chapter 6. Dynamic Programming, in *ALGORITHM DESIGN*, Addison Wesley, 2005, pp.251-335.
- [15] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving. New York, Addison-Wesley, 1983.
- [16] M. Likhachev, G. Gordon and S. Thrun, ARA*: Anytime A* with Provable Bounds on Sub-Optimality, Proceedings of IEEE international conference on neural information processing systems, vol. 16, 2003
- [17] R. E. Korf, Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, *Journal of Artificial Intelligence*, vol. 27, no. 1, pp. 97-109, 1985.
- [18] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report, 1998.
- [19] L.E. Kavraki, P. Svestka, J.-C. Latombe and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12 (4): 566-580, 1996.
- [20] P. Svestka and M. H. Overmars, Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, vol. 23, no. 3, 125--152, 1998.
- [21] A. t. Mors, C. Witteveen, J. Zutt and F. A. Kuipers. Context-Aware Route Planning. 8th German Conference, MATES 2010, Leipzig, Germany, pp. 138-149, 2010.

- [22] J.v.d. Berg, J. Snoeyink, M. Lin and D. Manocha, Centralized Path Planning for Multiple Agents: Optimal Decoupling into Sequential Plans, *Proc. Agentics: Science and Systems*, 2009.
- [23] W. Wang and W. B. Goh. Spatio-Temporal A* Algorithms for Offline Multiple Mobile Robot Path Planning. In Proceedings of 10th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 1091-1092, 2011.
- [24] W. Wang and W. B. Goh. Multi-robot Path Planning with the Spatio-Temporal A* Algorithm and Its Variants. AAMAS Workshops, 313-329, 2011.
- [25] M. Bennewitz, W. Burgard, S. Thrun. Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems. In Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 271-276, 2001.
- [26] D. Silver. Cooperative pathfinding. In AIIDE, pages 117-122, 2005.
- [27] N. Sturtevant and M. Buro. Improving Collaborative Pathfinding Using Map Abstraction. In Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 80-85, 2006.
- [28] Röger, G. and Helmert, M. (2012). Non-Optimal Multi-Agent Pathfinding Is Solved (Since 1984). Proceedings of the Fifth Annual Symposium on Combinatorial Search, 173-174.
- [29] Ryan, M. Multi-robot path planning with subgraphs. In Proc. of the 19th Australasian Conference Robotics and Automation, 2006.
- [30] Ryan, M. Graph Decomposition for Efficient Multi-robot Path Planning. Proceedings of the 20th international joint conference on Artificial intelligence, pp.2003-2008, 2007.
- [31] K.-H. C. Wang and A. Botea. Fast and Memory-Efficient Multi-Agent Pathfinding. Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, 2008.
- [32] K.-H. C. Wang and A. Botea. Tractable Multi-Agent Path Planning on Grid Maps. Proceedings of the International Joint Conference on Artificial Intelligence, 2009.

- [33] K.-H. C. Wang and A. Botea, MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees, *Journal of Artificial Intelligence Research*, 55-90, 2011.
- [34] M. Peasgood, C. M. Clark, and J. McPhee, A Complete and Scalable Strategy for Coordinating Multiple Agents Within Roadmaps, *IEEE TRANSACTION ON AGENTICS*, VOL.24, NO.2, 2008.
- [35] E. Masehian and A. H. Nejad, Solvability of Multi Agent Motion Planning Problems on Trees, *IEEE International Conference on Intelligent Agents and Systems*, 2009.
- [36] M. M. Khorshid, R. C. Holte and N. Sturtevant, A Polynomial-Time Algorithm for Non-optimal Multi-Agent Pathfinding, *Proceedings of the Fourth International Symposium on Combinatorial Search*, 2011.
- [37] P. Surynek. A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs. In *IEEE International Conference on Robotics and Automation*, pp.3613-3619, 2009.
- [38] P. Surynek. An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning. In *IEEE International Conference on Tools with Artificial Intelligence*, pp. 151-158, 2009.
- [39] R. Luna and K. E. Bekris. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp.294-300, 2011.
- [40] Q. Sajid, R. Luna and K. E. Bekris. Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives, *Proceedings of Fifth Symposium on Combinatorial Search*, 2012.
- [41] A. Krontiris, R. Luna and K. E. Bekris. *From Feasibility Tests to Path Planners for Multi-Agent Pathfinding. Symposium on Combinatorial Search*, 2013.
- [42] D. Ratner and M. K. Warmuth. Finding a Shortest Solution for the $N \times N$ Extension of the 15-PUZZLE Is Intractable. *National Conference on Artificial Intelligence*, 1986.

- [43] B. de Wilde. Cooperative Multi-Agent Path Planning. Master Thesis, Delft University of Technology.
- [44] J. Yu and S. M. LaValle. Time Optimal Multi-agent Path Planning on Graphs. *The First AAAI Workshop on Multiagent Pathfinding*, 2012.
- [45] W. Wang and W. B. Goh, An Iterative Approach For Makespan-Minimized Multi-Agent Path Planning in Discrete Space, in *Journal of Autonomous Agents and Multi-Agent Systems*. 2014.
- [46] Trevor Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. *In proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 173-178, 2010.
- [47] Trevor Standley and Richard Korf. Complete Algorithms for Cooperative Pathfinding Problems. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 668-673, 2011.
- [48] G. Wagner and H. Choset. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3260-3267, 2011.
- [49] Sharon et al., The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding, *proceedings of the twenty-second International Joint Conference on Artificial Intelligence*, vol. 1, 662—667, 2011.
- [50] G. Sharon, R. Stern, A. Felner and N. Sturtevant. Conflict-Based Search for Optimal Multi-Agent Path Finding, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- [51] M. B. Bazaraa, H. D. Sherali and C. M. Shetty. Chapter 10 Methods of Feasible Directions. In *Nonlinear programming, theory and algorithms*, 3rd, 537-654, 2006.
- [52] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly* 3: 95, 1956.

- [53] G. Zoutendijk, Methods of feasible directions: a study in linear and non-linear programming. 1960.
- [54] S. Boyd and L. Vandenberghe. Convex Optimization, Chapter 5. Duality. Cambridge University Press.
- [55] Dimitri P. Bertsekas, 5.1.1 Lagrange Multipliers, in Nonlinear Programming 2nd, Athena Scientific, Belmont, Massachusetts, 480-484, 1999.
- [56] J. Nocedal and S. J. Wright, 12.2 First Order Optimality Conditions, in Numerical Optimization, Springer, 1999.
- [57] J. Nocedal and S. J. Wright, 17.1 The Quadratic Penalty Method, in Numerical Optimization, Springer, 490-497, 1999.
- [58] J. Nocedal and S. J. Wright, 3. Line Search Methods, in Numerical Optimization, Springer, 34-63, 1999.
- [59] J. Nocedal and S. J. Wright, 4. Trust-Region Methods, in Numerical Optimization, Springer, 64-99, 1999.
- [60] Dimitri P. Bertsekas, 1.2 Gradient Method-Convergence, in Nonlinear Programming 2nd, Athena Scientific, Belmont, Massachusetts, 22-61, 1999.
- [61] Dimitri P. Bertsekas, 1.4 Newton's Method and Variations, in Nonlinear Programming 2nd, Athena Scientific, Belmont, Massachusetts, 88-101, 1999.
- [62] W.C. Davidon, Variable metric method for minimization. SIAM Journal on Optimization 1 (1), 1-17, 1991.
- [63] 50 Years of Integer Programming 1958-2008, Editors: M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi and L. Wolsey, Springer, 2010.
- [64] B. Korte and J. Vygen, Combinatorial Optimization Theory and Algorithms 3rd. Springer, 2006.

- [65] R. M. Karp, Reducibility Among Combinatorial Problems. Complexity of Computer Computations, page 85-103. Plenum Press, New York, 1972.
- [66] S. C. Brailsford, C. N. Potts and B. M. Smith, Constraint satisfaction problems: Algorithms and applications. European Journal of Operational Research 119, pp. 557-581, 1999.
- [67] D. S. Johnson and L. A. McGeoch, The traveling salesman problem: A case study in local optimization, Local search in combinatorial optimization, pp. 215-310, 1997
- [68] G. B. Mathews, On the partition of numbers. Proceedings of the London Mathematical Society 28: 486-490, 1897.
- [69] G. B. Dantzig and J. H. Ramser, The Truck Dispatching Problem, *Management Science* 6 (1): 80–91, 1959.
- [70] Robert J. Vanderbei. Linear Programming: Foundations and Extensions 3rd Edition, Chapter 2. Simplex Method. Springer, 2008.
- [71] J. Nocedal and S. J. Wright, Chapter 14 Linear Programming: Interior-Point Methods, in Numerical Optimization, Springer, 1999.
- [72] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica* 28(3). pp. 497–520, 1960.
- [73] M. Padberg and G. Rinaldi, A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *Siam Review*: 60–100, 1991.
- [74] J. Kleinberg and É. Tardos, Chapter 5. Divide and Conquer, in ALGORITHM DESIGN, Addison-Wesley, 209-250, 2005.
- [75] C. Voudouris, E. P.K. Tsang and A. Alsheddy. Guided Local Search, Handbook of Metaheuristics (Ed.2). (pp. 321-362). Springer, 2010.

- [76] H.R. Lourenço, O. Martin and T. Stützle. Iterated Local Search: Framework and Applications. *Handbook of Metaheuristics, 2nd. Edition*. Kluwer Academic Publishers, International Series in Operations Research & Management Science 146: 363–397, 2010.
- [77] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, Optimization by Simulated Annealing. *Science*, vol. 220, no. 4598, 1983.
- [78] P. Hansen, N. Mladenovic, J.A.M. Perez, Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175: 367–407, 2010.
- [79] T.A. Feo and M.G.C. Resende, Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.
- [80] D. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison Wesley, pp. 41, 1989.
- [81] J. R. Koza, Genetic programming: On the Programming of Computers by Means of Natural Selection, MIT Press Cambridge, MA, USA, 1992.
- [82] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. *Caltech Concurrent Computation Program* (report 826), 1989.
- [83] F. Glover, Heuristics for integer programming using surrogate constraints, *Decision Sciences*, 8:156-166, 1977.
- [84] Kennedy, J.; Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks IV*. pp. 1942–1948, 1995.
- [85] M. Dorigo, C. Blum, Ant colony optimization theory: A survey. *Theoretical Computer Science*, Vol. 344, No. 2-3, pp. 243-278, 2005
- [86] P. Rabanal, I. Rodríguez, and F. Rubio, Using River Formation Dynamics to Design Heuristic Algorithms, Springer, Volume 4618, pp. 163-177, 2007.

- [87] J.L. Zhou and A. L. Tits, Nonmonotone Line Search for Minimax Problems. Journal of optimization theory and applications, vol.76, no.3, 1993.
- [88] Z.B. Zhou, X. Cai, J. Jian, An Improved SQP algorithm for solving minimax problems, Applied Mathematics Letters 22, 464-469, 2009.
- [89] F. Ye, H. Liu, S. Zhou, S. Liu, A smoothing trust-region Newton-CG method for minimax problem. Journal of Applied Mathematics and Computation, vol.199, issue 2, pp.581-589, 2008.
- [90] F.S. Wang, K.C. Zhang, A hybrid algorithm for minimax optimization, Annals of Operations Research 164, 67-191, 2008.
- [91] F. Wang and Y. Wang, Nonmonotone algorithm for minimax optimization problems. Applied Mathematics and Computation, 2011.
- [92] C. Charalambous and A. R. Conn, An Efficient Method to Solve the Minimax Problem Directly, SIAM Journal on Numerical Analysis, Vol. 15, no. 1, pp 162-187, 1978.
- [93] A. Vardi, New Minimax Algorithm. Journal of Optimization Theory and Applications, Vol. 75, no. 3, 1992.
- [94] G. Di Pillo and L. Grippo, A smooth method for the finite minimax problem, Mathematical Programming 60, 187-214, 1993.
- [95] X. Li, An Entropy-Based Aggregate Method for Minimax Optimization, Engineering Optimization, vol. 18, pp. 277-285, 1997.
- [96] J. Li and J. Huo, Inexact smoothing method for large scale minimax optimization. Journal of Applied Mathematics and Computation, vol. 218, pp. 2750-2760, 2011.
- [97] J-P. Watson, (2010). An Introduction to Fitness Landscape Analysis and Cost Models for Local Search. In M. Gendreau, J.-Y. Potvin, Handbook of Metaheuristics(Ed.2). (pp. 599-623). Springer.

- [98] F. Glover and M. Laguna, Modern Heuristic Techniques for Combinatorial Optimization, Chapter 3: Tabu search, Blackwell Scientific Publications, 1993.
- [99] T. Weise, Global Optimization Algorithms-Theory and Application, Germany: it-weise.de (self-published), 2009. [Online]. Available: <http://www.it-weise.de/>.
- [100] J. Yu, S. M. LaValle, Planning optimal paths for multiple robots on graphs. Retrieved from http://msl.cs.uiuc.edu/jyu18/_mapp.html on Nov 15th, 2013.
- [101] B. Bazaraa, H. D. Sherali and C. M. Shetty. Chapter 9.1 Concept of Penalty functions. In Nonlinear programming, theory and algorithms, 3rd, 470-475, 2006.
- [102] X. He, A dual algorithm for minimax problems, Journal of Applied Mathematics and Computing, Vol. 17, issue 1-2, pp. 401-418, 2005.
- [103] Y. H. Yu and L. Gao. Nonmontone Line Search Algorithm for Constrained Minimax Problems. Journal of Optimization Theory and Applications, vol. 115, issue 2, pp. 419-446, 2002.
- [104] S. Yu, Smoothing Method for Minimax Problems, Computational Optimization and Applications, vol. 20, issue 3, pp.267-279, 2001.
- [105] J. Ma and X. Zhang, Pattern search methods for finite minimax problems, Journal of Applied Mathematics and Computing, vol. 32, issue 2, pp.491-506, 2010.
- [106] R. T. Marler and J. S. Arora, Survey of multi-objective optimization methods for engineering. Struct Multidisc Optimization 26, pp. 369-395, 2004.
- [107] D. Parr and J. M. Thompson, Solving the multi-objective nurse scheduling problem with a weighted cost function, Annals of Operation Research, vol. 155, issue 1, pp. 279-288, 2007.
- [108] C. Bishop, Chapter 11.1.2 Rejection sampling, in Pattern recognition and machine learning, 528-530, 2007.
- [109] R. M. Neal, Slice sampling, Annals of Statistics, vol. 31, no. 3, 705-767, 2003.

- [110] P. Hansen and N. Mladenović, First vs. Best Improvement: An Empirical Study, in Journal of Discrete Applied Mathematics-Special issue: IV ALIO/EURO workshop on applied combinatorial optimization, vol. 154, issue 5, pp. 802-817, 2006.
- [111] H. H. Hoos and T. Stützle, Chapter 1.4 Search Paradigms, in Stochastic Local Search: Foundation and Application, pp.31-37, 2005.
- [112] V. Pereyra, M. Saunders and J. Castillo. Equispaced Pareto front construction for constrained bi-objective optimization. Mathematical and Computer Modelling, vol. 57, issue 9-10, pp. 2122-2131, 2013.
- [113] I. Y. Kim and O.L. de Weck, Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. Structural and Multidisciplinary Optimization, vol. 31, issue 2, pp. 105-116, 2006.
- [114] M. Ehrgott, Chapter 2. Efficiency and Nondominance, in Multicriteria Optimization 2th Edition, Springer, 2005.
- [115] I. Das and J. E. Dennis, Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization, vol. 8, issue 3, pp. 631-657, 1998.
- [116] A. Messac, A. Ismail-Yahaya, C.A. Mattan, The normalized normal constraint method for generating the Pareto frontier, Structural and Multidisciplinary Optimization, vol. 25, issue 2, pp. 86-98, 2003.
- [117] T. Erfani and S. V. Utyuzhnikov, Directed Search Domain: A Method for Even Generation of Pareto Frontier in Multiobjective Optimization, in Journal of *Engineering Optimization*, vol. 43, no. 5, pp. 1-18, 2011.
- [118] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, 2002.

- [119] N. Beume, B. Naujoks and M. Emmerich, SMS-EMOA: Multiobjective selection based on dominated hypervolume, in *European Journal of Operational Research*, vol. 181, issue 3, pp. 1653-1669, 2007.
- [120] S. Bandyopadhyay and U. Maulik, A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA, *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, 2008.
- [121] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, MIT Press, 1998.
- [122] G.W. Brown, Iterative Solutions of Games by Fictitious Play, in *Activity Analysis of Production and Allocation*, T.C. Koopmans (Ed.), New York: Wiley, 1951.

APPENDIX A1. Lemma 1 and Lemma 2

Lemma 1. Assume $x \in \mathbf{R}^+$, $y \in \mathbf{Z}^+$, μ is set to a sufficiently high value such that $\mu > x_i * f_i(r)$ for $r \in N_i(r^{cur})$ in the function (27). For any two solutions $r^{(a)}$ and $r^{(b)}$ in $N_i(r^{cur})$, $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$ only if $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$ or $f_i(r^{(a)}) < f_i(r^{(b)})$ given $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$.

Proof. For any two solutions $r^{(a)}$ and $r^{(b)}$ in $N_i(r^{cur})$:

(a) If $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$,

$$\begin{aligned} F_{q,i}(r^{(b)}; \mu, x, y) &= x_i * f_i(r^{(b)}) + \mu * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\} \\ &\geq \{x_i * f_i(r^{(b)})\} + \mu * [\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} + 1] \\ &> \{x_i * f_i(r^{(b)})\} + \mu * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} + \{x_i * f_i(r^{(a)})\} \\ &\geq F_{q,i}(r^{(a)}; \mu, x, y) \end{aligned} \tag{A1.1}$$

So $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$ always hold if $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$.

(b) if $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$,

$$\begin{aligned} F_{q,i}(r^{(b)}; \mu, x, y) &= \{x_i * f_i(r^{(b)})\} + \mu * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\} \\ &= \{x_i * [f_i(r^{(b)}) - f_i(r^{(a)})]\} + F_{q,i}(r^{(a)}; \mu, x, y) \end{aligned} \tag{A1.2}$$

So given $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$, $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$ only if $f_i(r^{(a)}) < f_i(r^{(b)})$.

(c) if $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} > \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$, according to (A1.1),

$$F_{q,i}(r^{(a)}; \mu, x, y) > F_{q,i}(r^{(b)}; \mu, x, y) \quad \text{A1.3}$$

In summary, $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$ only if $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$ or $f_i(r^{(a)}) < f_i(r^{(b)})$ given $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$.

Lemma 2. Assume $\eta_i = \eta^{x_i}$, $\beta_{i,j} = \beta_i^{y_{i,j}}$ and $\beta_i \leq \eta^\mu$ where $i=1,2,\dots,n$, $j=1,2,\dots,n$, $i \neq j$, $x \in \mathbf{R}^+$, $y \in \mathbf{Z}^+$ and μ is set to a sufficiently high value such that $\mu > x_i * f_i(r)$ for $r \in N_i(r^{cur})$ in the function $F_{q,i}(r; \mu, x, y)$ given in (27), then $P_s(r_i | r_{-i}^{cur})$ in (64) is strictly decreasing with the function value of $F_{q,i}(r; \mu, x, y)$ when $\eta \in (0,1)$ and $\beta_i \in (0,1)$, and non-increasing with the function value of $F_{q,i}(r; \mu, x, y)$ when $\eta \in (0,1]$ and $\beta_i \in [0,1]$.

Proof. For any two solutions $r^{(a)}$ and $r^{(b)}$ in $N_i(r^{cur})$ such that $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$, according to **Lemma 1**, $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$ or $f_i(r^{(a)}) < f_i(r^{(b)})$ given $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$.

When $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$,

(a) If $\eta \in (0,1)$, then $\beta_i \in [0,1)$ and $\beta_i \leq \eta^\mu$ can be transformed into the following equation

$$\eta^{\mu+\Delta} = \beta_i, \text{ where } \Delta \geq 0 \text{ and } \Delta = +\infty \text{ if } \beta_i = 0 \quad \text{A1.4}$$

So $P_s(r_i | r_{-i}^{cur})$ can be expressed as follow

$$P_s(r_i | r_{-i}^{cur}) = \eta^{x_i * f_i(r) + (\mu + \Delta) * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r)\}} \quad \text{A1.5}$$

Since $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$, so

$$x_i * f_i(r^{(a)}) + (\mu + \Delta) * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} < x_i * f_i(r^{(b)}) + (\mu + \Delta) * \quad \text{A1.6}$$

$$\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\} \text{ if } \Delta < +\infty$$

While

$$x_i * f_i(r^{(a)}) + (\mu + \Delta) * \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} \leq x_i * f_i(r^{(b)}) + (\mu + \Delta) * \quad \text{A1.7}$$

$$\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\} \text{ if } \Delta = +\infty$$

Thus

$$P_s(r_i^{(a)} | r_{-i}^{cur}) > P_s(r_i^{(b)} | r_{-i}^{cur}) \text{ if } \eta \in (0,1) \text{ and } \beta_i \in (0,1) \quad \text{A1.8}$$

And

$$P_s(r_i^{(a)} | r_{-i}^{cur}) \geq P_s(r_i^{(b)} | r_{-i}^{cur}) \text{ if } \eta \in (0,1) \text{ and } \beta_i = 0 \quad \text{A1.9}$$

(b) If $\eta=1$, then $\beta_i \in [0,1]$,

$$P_s(r_i | r_{-i}^{cur}) = \beta_i^{\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r)\}} \quad \text{A1.10}$$

So

$$P_s(r_i^{(a)} | r_{-i}^{cur}) > P_s(r_i^{(b)} | r_{-i}^{cur}) \text{ if } \eta=1 \text{ and } \beta_i \in (0,1) \quad \text{A1.11}$$

And

$$P_s(r_i^{(a)} | r_{-i}^{cur}) \geq P_s(r_i^{(b)} | r_{-i}^{cur}) \text{ if } \eta=1 \text{ and } \beta_i = 0 \text{ or } 1 \quad \text{A1.12}$$

When $\sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(a)})\} = \sum_{j=1, j \neq i}^n \{y_{i,j} * \Gamma_{i,j}(r^{(b)})\}$ and $f_i(r^{(a)}) < f_i(r^{(b)})$:

a) if $\eta \in (0,1)$,

$$P_s(r_i^{(a)}|r_{-i}^{cur}) > P_s(r_i^{(b)}|r_{-i}^{cur}) \text{ if } \eta \in (0,1) \text{ and } \beta_i \in (0,1) \quad \text{A1.13}$$

And

$$P_s(r_i^{(a)}|r_{-i}^{cur}) \geq P_s(r_i^{(b)}|r_{-i}^{cur}) \text{ if } \eta \in (0,1) \text{ and } \beta_i=0 \quad \text{A1.14}$$

b) if $\eta=1$,

$$P_s(r_i^{(a)}|r_{-i}^{cur}) \geq P_s(r_i^{(b)}|r_{-i}^{cur}) \text{ if } \eta=1 \text{ and } \beta_i \in [0,1] \quad \text{A1.15}$$

In summary, for any two solutions $r^{(a)}$ and $r^{(b)}$ in $N_i(r^{cur})$ such that $F_{q,i}(r^{(a)}; \mu, x, y) < F_{q,i}(r^{(b)}; \mu, x, y)$, $P_s(r_i^{(a)}|r_{-i}^{cur}) > P_s(r_i^{(b)}|r_{-i}^{cur})$ when $\eta \in (0,1)$ and $\beta_i \in (0,1)$, while $P_s(r_i^{(a)}|r_{-i}^{cur}) \geq P_s(r_i^{(b)}|r_{-i}^{cur})$ when $\eta \in (0,1]$ and $\beta_i \in [0,1]$.

APPENDIX A2. Proofs of Theorem 1 and Theorem 2

Theorem 1. If **Condition 1** holds, $P_{\eta,\beta}(r|r^{cur})$ in (67) for $r \in N_i(r^{cur})$ is strictly decreasing with the function value of $F_{q,i}(r; \mu, x, y)$ in (27).

Proof. Since the solution r in $N_i(r^{cur})$ with lower $F_{q,i}(r^{(b)}; \mu, x, y)$ is given the higher priority trial order, then for any two solutions $r^{(m)}$ and $r^{(m+1)}$ ($m=1,2,\dots$) in $N_i(r^{cur})$, $F_{q,i}(r^{(m)}; \mu, x, y) < F_{q,i}(r^{(m+1)}; \mu, x, y)$. If **Condition 1** holds, according to **Lemma 2** in Appendix A1,

$$P_s(r_i^{(m)}|r_{-i}^{cur}) > P_s(r_i^{(m+1)}|r_{-i}^{cur}) \quad A2.1$$

As $P_s(r^{(m)}|r_{-i}^{cur}) > 0$ when $\eta \in (0,1)$ and $\beta_i \in (0,1)$, so

$$\prod_{j=1}^{m-1} (1 - P_s(r_i^{(j)}|r_{-i}^{cur})) > \prod_{j=1}^m (1 - P_s(r_i^{(j)}|r_{-i}^{cur})) \quad A2.2$$

Then

$$P_g(r_i^{(m)}|r_{-i}^{cur}) > P_g(r_i^{(m+1)}|r_{-i}^{cur}) \quad A2.3$$

As a result, $P_g(r_i|r_{-i}^{cur})$ is strictly decreasing with $F_{q,i}(r; \mu, x, y)$ if **Condition 1** holds. As $P_{\eta,\beta}(r|r^{cur}) \propto P_g(r_i|r_{-i}^{cur})$ and therefore $P_{\eta,\beta}(r|r^{cur})$ is strictly decreasing with the function value of $F_{q,i}(r; \mu, x, y)$ if **Condition 1** holds.

Theorem 2. If **Condition 2** holds, $P_{\eta,\beta}(r|r^{cur})$ in (67) for $r \in N_i(r^{cur})$ is non-increasing with the function value of $F_{q,i}(r; \mu, x, y)$ in (27).

Proof. Since the solution r in $N_i(r^{cur})$ with lower $F_{q,i}(r; \mu, x, y)$ is given the higher priority trial order, then for any two solutions $r^{(m)}$ and $r^{(m+1)}$ ($m=1,2,\dots$) in $N_i(r^{cur})$,

$F_{q,i}(r^{(m)}; \mu, x, y) < F_{q,i}(r^{(m+1)}; \mu, x, y)$. If **Condition 2** holds, according to **Lemma 2** in appendix A1,

$$P_s(r_i^{(m)} | r_{-i}^{cur}) \geq P_s(r_i^{(m+1)} | r_{-i}^{cur}) \quad \text{A2.4}$$

As $P_s(r^{(m)} | r_{-i}^{cur}) \geq 0$ when $\eta \in (0,1]$ and $\beta_i \in [0,1]$, so

$$\prod_{i=1}^{m-1} (1 - P_s(r_i^{(i)} | r_{-i}^{cur})) \geq \prod_{i=1}^m (1 - P_s(r_i^{(i)} | r_{-i}^{cur})) \quad \text{A2.5}$$

Then

$$P_g(r_i^{(m)} | r_{-i}^{cur}) \geq P_g(r_i^{(m+1)} | r_{-i}^{cur}) \quad \text{A2.6}$$

As a result, $P_g(r_i | r_{-i}^{cur})$ is non-increasing with $F_{q,i}(r; \mu, x, y)$ if **Condition 2** holds. As $P_{\eta,\beta}(r | r^{cur}) \propto P_g(r_i | r_{-i}^{cur})$ and therefore $P_{\eta,\beta}(r | r^{cur})$ is non-increasing with the function value of $F_{q,i}(r; \mu, x, y)$ if **Condition 2** holds.