A Stochastic Algorithm for Makespan Minimized Multi-Agent Path Planning in Discrete Space

Article *in* Applied Soft Computing · February 2015

DOI: 10.1016/j.asoc.2015.01.046

2 authors, including:

Boon Wooi Goh
Nanyang Technological University
57 PUBLICATIONS **356** CITATIONS

Available online at www.sciencedirect.com
ScienceDirect
Applied Soft Computing 00 (2014) 1–26
Applied Soft Computing

A Stochastic Method for Makespan Minimized Multi-Agent Path Planning in Discrete Space

Wenjie Wang*, Wooi Boon Goh

School of Computer Engineering, Nanyang Technological University, Singapore

Abstract

Makespan minimized Multi-Agent Path Planning (MAPP) requires the minimization of the time taken by the slowest agents to reach its destination. The resulting minimax objective function is non-smooth and the search for an optimal solution in MAPP can be intractable. In this work, a maximum entropy function is adopted to approximate the minimax objective function. A stochastic algorithm named Probabilistic Iterative Makespan Minimization (PIMM) is then proposed to find a makespan minimized MAPP solution by solving a sequence of computationally simpler MAPP minimization problems. At each iteration, a novel local search algorithm called Probabilistic Iterative Path Coordination (PIPC) is used to find a sufficiently good solution for each MAPP minimization problem. Experimental results from comparative studies with existing MAPP algorithms show that the proposed algorithm strikes a good tradeoff between the quality of the makespan minimized solution and the computational cost.

Keywords:
path planning, stochastic local search

1. Introduction

Multi-Agent Path Planning (MAPP) requires finding a collision-free route r_i for each agent A_i that navigates the agent A_i from its initial position to its unique destination. The MAPP problem arises in practical applications such as path planning of unmanned autonomous vehicles on a factory floor or route planning for a fleet of delivery trucks in a city. The traditional approaches for solving the MAPP problem can be mainly divided into coupled or decoupled approach. The coupled approaches simultaneously plan the paths for all agents and can therefore find an optimal solution. However, finding this optimal solution creates a scalability issue as the problem can become computationally intractable as the number of agents or the size of the search map increases. In contrast, the decoupled approach decomposes the MAPP problem into several sub-problems, and then solves each sub-problem separately. Though the computational efficiency of decoupled approaches can be improved by reducing the dimensionality of the search space, this often sacrifices the optimality of the solution and may make this approach incomplete.

In this paper, we are interested in addressing the makespan minimized MAPP problem in a grid map. In the scenario where all n agents move concurrently, this can effectively be viewed as the problem of minimizing the time related cost of the agent that takes the maximum time to reach its destination. This is essentially a minimax optimization problem, which is previously solved by line search approaches ([1], [2]) or trust region methods ([3], [4],

*Corresponding author
Email address: wang0570@e.ntu.edu.sg (Wenjie Wang)

[5]). This work adopts a maximum entropy function [6] to approximate the minimax objective function and proposed a stochastic algorithm named Probabilistic Iterative Makespan Minimization (PIMM) to approximate the optimal makespan solution by solving a sequence of computationally simpler MAPP minimization problems. However, finding a globally optimal solution for a MAPP problem can be intractable ([7], [8]). In this case, a novel local search algorithm called Probabilistic Iterative Path Coordination (PIPC) for MAPP is used to find a sufficiently good solution for each MAPP sub-problem. The proposed PIPC algorithm incorporated a unique stochastic mechanism into the local search method. In doing so, it probabilistically generates a neighboring solution and therefore avoids being trapped in local optima. By providing informed tuning strategies, PIPC is able to efficiently switch between extensive and intensive search in the solution space, as a result, it makes a computationally efficient performance to find a sufficient good solution.

This paper is organized as follows: section 2 discusses some previous related work in MAPP, followed by section 3, which provides a formal definition of the MAPP problem that is being addressed in this work. The detail of the proposed stochastic local search algorithm is described in section 4. Experimental results and discussions are presented in section 5, followed by a conclusion in section 6.

2. Related Work

Since finding an optimal MAPP solution can be intractable, previously proposed approaches for Multi-Agent Path Planning often made a choice between optimality of the solution and computational efficiency of the algorithm.

Computationally efficient MAPP algorithms like decoupled approach reduce the size of search space by decomposing the MAPP problem into several sub-problems and therefore may quickly find a solution, but sub-optimal. One type of decoupled approach called prioritized planning 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. [12] 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. [11] 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 [13] 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. [14] 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 [15] 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 [16] 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*.

Another type of decoupled approach called path coordination first finds an individual sub-solution for each sub-problem independently and then resolve the conflicts between the sub-solutions by using some coordination schemes. Ryan [17] 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 [18]. 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 [19] 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 [23] 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 [24] 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. [20] 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 [21] 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. [22] 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 [34] proposed the BIBOX algorithm for MAPP in bi-connected environments with at least two unoccupied vertices. He later proposed BIBOX-θ [35], 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 [25] 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 swapped the position with the agent ahead and kept the positions of other agents unchanged. Sajid et al. [26] recently modified the Push and Swap algorithm into a parallel version that allows the concurrent movement of agents. Krontiris et. al [38] 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 [33]. Wilde claimed in [31] 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.

Optimal MAPP algorithms are able to find an optimal solution, but they unfortunately suffered from a rapidly increasing search space with the number of agent and the size of the map. Svestka and Overmars [9] proposed a super graph to plan the paths for all agents in a coupled way. In this graph, each adjacent super node is reached by moving only one of the agents in a probabilistically generated map. Unfortunately, their algorithm is only probabilistically complete due to their use of a probabilistic roadmap planner. Moreover, their algorithm has an exponentially increasing search space as the number of agent increases. Standley [28] 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* 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 [29] went on to modified 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 [36] 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. [10] 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. [27] later proposed another optimal approach that solves the MAPP sub-problems iteratively 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 reviewed optimal MAPP algorithms guarantee global optimality, they unfortunately still suffer from rapidly increasing search space as the number of agents or the map size increases. In short, finding a global optimal solution in MAPP can be intractable.

A specific goal of this work is to find makespan-minimized MAPP solutions. From this perspective, a closely related work is that of Yu and LaValle [37]. They 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. Like many other optimal MAPP algorithms, their ILP algorithm suffers from the rapidly increasing search space with the number of agents. In a crowded environment with a large number of agents, the computational time taken to return an optimal feasible solution may become unacceptable. In this paper, a novel computationally efficient anytime algorithm for makespan-minimized MAPP is proposed. By addressing only a single-agent path planning sub-problem within each iteration of the algorithm, we have avoided the issue of rapidly increasing search space as the number of agents increase.

3. Makespan Minimized Multi-Agent Path Planning Problem Definition

The MAPP problem for n agents is commonly described as the problem of finding a feasible path r_i for each agent A_i, where $i = 1, 2, ..., n$, so that each agent can move from a starting point $I_i = (x_{I_i}, y_{I_i})$ to its unique destination $G_i = (x_{G_i}, y_{G_i})$ without colliding with other agents or static obstacles S. Since the MAPP problem is considered to be completed when all the agents reach their destinations, the additional goal addressed in this paper is to find a makespan minimized solution which has the minimal task completion time. More specifically, the makespan minimized MAPP problem addressed here is defined as follows: 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.

In this work, the environment is modeled on a four connected undirected grid map and time is discretized into a series of unit time steps. The assumption that all mobile agents are of unit size allows us to directly map the starting and destination locations of each agent to the vertices in the grid map. At each unit time step, choosing one of the four possible moving actions (i.e. North, South, East, West) 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. For simplicity, a rotation action is assumed unnecessary as the agents can freely and instantaneously traverse in any one of the four perpendicular directions at each node. The position of each agent A_i for $i \in \{1, ..., n\}$ 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 $(r_i(0), ..., r_i(t), ..., r_i(T_i))$ where $r_i(0) = (x_{I_i}, y_{I_i}, 0)$ 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, we define these paths with the same trajectory in space and time but different total time step T_i as the equivalent paths.

Definition 1. *Equivalent paths - Two different paths r_i^a and r_i^b for $i = 1, 2, ..., n$, are equivalent if $r_i^a(t) = r_i^b(t)$ for $t \in \{1, ..., min(T_i^a, T_i^b)\}$ 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 $t \in \{min(T_i^a, T_i^b) + 1, ..., max(T_i^a, T_i^b)\}$.*

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 $i = 1, 2, ..., n$, are the same or equivalent path.*

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

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

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

$$r_i(t) = L(S_k) \text{ for each } \{i, k, t\} \tag{2}$$

where $i \in \{1, ..., n\}, k \in \{1, ..., m\}, t \in \{1, ..., T_i\}$ and $L(S_k)$ denotes the position of the static obstacle S_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

$$\{r_i(t) = r_j(t-1) \text{ and } r_i(t-1) = r_j(t)\} \text{ for each } \{i, j, t\} \tag{3}$$

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

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 (1), (2) and (3) respectively. In order to guarantee a feasible solution, the following conditions are required to be satisfied:

$$\varphi_i(t) = 0 \text{ for each } \{i, t\} \tag{4}$$

$$\Omega_{i,j}(t) = 0 \text{ and } \phi_{i,j}(t-1, t) = 0 \text{ for each } \{i, j, t\} \tag{5}$$

where $i \in \{1, ..., n\}$, $j \in \{1, ..., n\}$, $j \neq i$ and $t \in \{1, ..., T_{i,j}\}(T_{i,j} = max(T_i, T_j))$. The constraints in (4) and (5) are divided into two types, namely *static constraint* and *dynamic constraint*. In this work, the static constraints are strictly satisfied. In feasible problem instances, removing the vertices occupied by static obstacles does not destroy the connectedness of the grid map, as such, setting the static obstacles strictly satisfied avoids unnecessary search for the space occupied by static obstacles instead. When $T_i < T_j$, the collision detection 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 (4) and it is not time related.*

Definition 4. *Dynamic constraint - is defined as a collision constraint imposed by the incremental paths of agents in function (5) at any given time step.*

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

Definition 5. *Free node v - is defined a path node that satisfies both the collision constraints given in functions (4) and (5), and can therefore be freely traversed by an agent.*

Definition 6. *Collision node $\bar{v}$ - is defined as a path node that satisfies only the collision constraints given in function (4) and may be traversed by an agent under some conditions..*

Since each path consists of a sequence of path nodes, we then can define the feasible path, feasible solution and feasible region as follows:

Definition 7. *Feasible path r_i - is a path consisting of only free nodes.*

Definition 8. *Feasible solution $r = (r_1, ..., r_n)$ - is defined as a solution in which each path r_i for $i = 1, 2, ..., n$ 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 (4) can (5).*

Assume the objective function $f_i(r)$ for $i \in 1, ..., n$ measures the time related traversing cost of r_i, then the makespan minimized MAPP problem is mathematically described as

$$\min_{r} \ F_0(r) = \max_{1 \le i \le n} f_i(r) \tag{6}$$

Subjected to (4) and (5). Due to the unspecified T_i for each agent A_i, many equivalent solutions with equal $F_0(r)$ exist in the solution space, so it is possible to reduce the size of search space by eliminating the redundant equivalent solutions as given in Definition 2. This will be further discussed in the next section. The following section will introduce a stochastic algorithm for the makespan minimized MAPP problem.

4. The Stochastic Algorithm for Makespan Minimized Multi-Agent Path Planning

The objective function $F_0(r)$ in (6) is a non-smooth function, in this work we approximate the minimax objective function by using a maximum entropy function given by

$$F_e(r; c) = (1/c) \ln\{\sum_{i=1}^{n} \exp(c * f_i(r))\} \tag{7}$$

As $c \to +\infty$, $F_e(r; c) \to F_0(r)$. As a result, the approximate problem of the minimax problem (6) is given by

$$\min_{r} \ F_e(r; c) = (1/c) \ln\{\sum_{i=1}^{n} \exp(c * f_i(r))\} \tag{8}$$

Subjected to (4) and (5). An incremental scheme is recommended for the control parameter c, in which c is initially set to a small value and then gradually increased. In so doing, the optimal solution of the minimax problem (6) can be approximated by solving a series of minimization problem (8) with gradually increasing c. The minimization problem (8) can be solved by feasible direction method [39], which moves to an improving feasible solution from the current feasible solution along a descent feasible direction d. As such, for any two iterated 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 (8) can be eventually obtained as the iteration k continues. To use feasible direction method in the minimization problem (8), 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 (4), (5) and the following condition:

$$\nabla F_e(r; c) * d < 0 \tag{9}$$

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

$$\nabla F_e(r; c) = \sum_{i=1}^{n} \{[1/\sum_{j=1}^{n} \exp(c * [f_j(r) - f_i(r)])] * \vec{f_i}\} \tag{10}$$

where $\vec{f_i}$ is the coordinate vector associated with the variable $x_i = f_i(r)(i \in \{1, ..., n\})$. As a result, at the iteration k of the feasible direction approach for the minimization problem (8), given the current feasible solution $r^{(k-1)}$, an new solution r' is considered as a possible improving feasible solution when it satisfy the constraints (4) and (5) and the following condition

$$\sum_{i=1}^{n} \{[1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])] * f_i(r')\} < \sum_{i=1}^{n} \{[1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])] * f_i(r^{(k-1)})\} \tag{11}$$

The next issue is selecting an improving feasible solution as the iterated 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

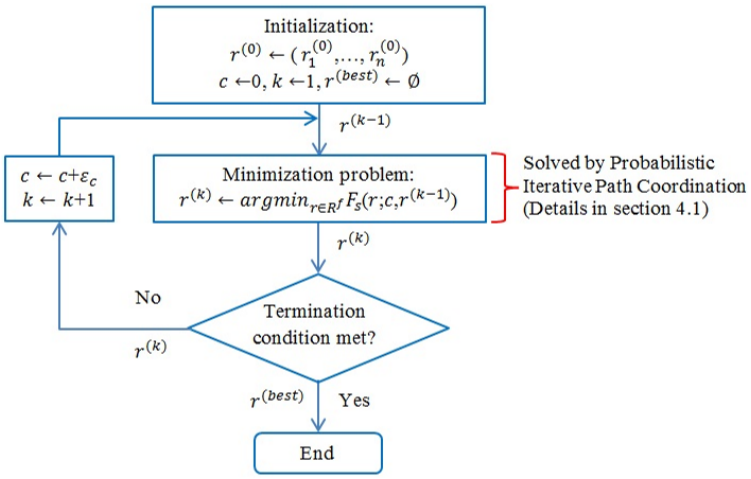

Figure 1. Overview description of the Probabilistic Iterative Makespan Minimization (PIMM) algorithm.

be intractable. Alternatively, one could just find an improving feasible solution near the last iterated 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, at the iteration k, the iterated solution $r^{(k)}$ can be obtained by solving the following problem with a local search method once the condition (11) is satisfied.

$$\min_r \; F_s(r;c,r^{(k-1)}) = \sum_{i=1}^{n}\{[1/\sum_{j=1}^{n}\exp(c*[f_j(r^{(k-1)})-f_i(r^{(k-1)})])]*f_i(r)\} \tag{12}$$

subjected to (4) and (5). A problem in solving the minimization problem (12) is that the starting solution $r^{(1)}$ and the iterated solutions $r^{(k)}$ are required to be feasible. However, this is a challenging problem in crowded environments. For the iterated solutions, though a basic local search method is capable of finding an improving solution in the local neighborhood, increasing dynamic constraints with the increasing agent number quickly decreases the probability that feasible solutions exist in the local neighborhood. As a result, an unfeasible solution may be returned by the basic local search method. To address this problem, a novel stochastic local search algorithm called *Probabilistic Iterative Path Coordination* (PIPC) is proposed to find an improving feasible solution $r^{(k)}$ at each iteration k. A good starting feasible solution $r^{(1)}$ can be obtained by using PIPC to solve the minimization problem (12) with $c = 0$ and a given initial solution $r^{(0)}$. Here the initial solution $r^{(0)}$ does not need to be feasible. In our implementation, $r^{(0)}$ is obtained by planning an individual path for each agent with all other agents assumed absent.

In summary, as shown in Figure 1, given an initial solution $r^{(0)}$, the iterative algorithm solves a sequence of minimization problem (12) defined by gradually increasing c from 0 until some stopping criteria are satisfied. Due to the nondeterministic nature of PIPC and the possibility that no feasible solution exists, the iterative algorithm is made terminate when a preset time bound is reached or the current best solution $r^{(best)}$ has not been improved at all after δ iterations in PIPC. In our implementation, δ is set to $15 * n$, where n is the number of agents. We call the proposed iterative algorithm for makespan minimized MAPP *Probabilistic Iterative Makespan Minimization* (PIMM).

4.1. Probabilistic Iterative Path Coordination Algorithm

The basic local search method iteratively moves from the current solution r^{cur} to an improving neighbor solution in a neighborhood $N(r^{cur})$ 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 things needed to be determined a priori, namely the local neighborhood, the navigation strategy and the cost function.

Definition 10. *The move operator [32] specifies the allowable modifications to one component r_i of the current solution $r^{cur} = (r_1^{cur}, ..., r_n^{cur})$.*

Definition 11. *The neighborhood $N(r^{cur})$ of the current solution $r^{cur} = (r_1^{cur}, ..., r_n^{cur})$ contains all the solutions that can be reached by executing one of the move operators once. Note that the current solution r^{cur} is also included in its own neighborhood.*

The definition of the local neighborhood has a significant influence on the performance of the local search method. A large neighborhood may result in less number of locally optimal solutions, but is computationally costly to evaluate. On the contrary, a small neighborhood is more computationally efficient, but produces more locally optimal solutions. In this work, the move operator is defined in definition 10. In doing so, n different neighborhoods $N_1(r^{cur}), ..., N_n(r^{cur})$ are provided for each solution r^{cur}, and each neighborhood $N_i(r^{cur})$ includes all the solutions obtained by enumerating all changes in one component r_i of the current solution r^{cur}. This is because an optimal path can be found from $N_i(r^{cur})$ by computationally efficient shortest path algorithms like the Dijkstra or A* algorithm.

Definition 12. *The solution r^* is called a locally optimal solution if $f(r^*) \leq f(r')$ for any $r' \in \{N_1(r^*) \cup ... \cup N_n(r^*)\}$, where f is the cost function.*

Definition 13. *The solution r^* is called a globally optimal solution if $f(r^*) \leq f(r')$ for any r' in the entire solution space, where f is the cost function.*

The navigation strategy indicates the selection of the next iterated solution. Several navigation strategies can be used here, namely first improvement, best improvement and so forth. Since a better solution will speed up convergence of the algorithm to an optimal solution, so we adopt the best improvement strategy and select the best solution in the neighborhood $N_i(r^{cur})$.

The final cost function indicates the solution quality and it can be derived from the objective function in the original problem. However, in a constrained optimization problem like that given in (12), additional constraints need to be satisfied. As such, at each iteration the improving neighbor solution is obtained with respect to the objective function $F_s(r; c, r^{(k-1)})$ and the constraints (4) and (5). A common way is to relax the constraints by adding a penalty term to the objective function. This penalty term measures the violations of the constraints. In this work, the dynamic constraints (5) are relaxed and the cost function is given by

$$F_p(r;\mu,c,r^{(k-1)}) = \sum_{i=1}^{n}\{[1/\sum_{j=1}^{n}\exp(c*[f_j(r^{(k-1)})-f_i(r^{(k-1)})])]*f_i(r)\}+\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\sum_{t=1}^{T_{i,j}}\{0.5*\mu*[\Omega_{i,j}(t)+\phi_{i,j}(t-1,t)]\} \quad (13)$$

where μ is a non-negative penalty coefficient. In summary, beginning with an initial solution $r^{[0]}$, the basic local search approach obtains each subsequent solution $r^{[\tau]}(\tau = 1, 2, ...)$ by selecting the best solution from $N_i(r^{[\tau-1]})$ in a systematic manner with respect to the cost function in (13) until no further improvement is possible. Since all the solutions in each neighborhood $N_i(r^{[\tau-1]})$ only differ in the path assignments for one component r_i of the current solution $r^{[\tau-1]}$, at each iteration τ the comparison among the solutions in $N_i(r^{[\tau-1]})$ can be done by only evaluating the cost of the path r_i instead of the whole solution r. With this, an improving solution can be easily and quickly determined from $N_i(r^{[\tau-1]})$ by solving a single agent path planning problem. In this single agent path planning problem, based on the incremental paths of all other agents and static obstacles, the agent A_i can find an optimal path r_i with respect to the cost function given by

$$F_{p,i}(r;\mu,c,r^{(k-1)}) = \{[1/\sum_{j=1}^{n}\exp(c*[f_j(r^{(k-1)})-f_i(r^{(k-1)})])]*f_i(r)\}+\sum_{j=1,j\neq i}^{n}\sum_{t=1}^{T_{i,j}}\{\mu*[\Omega_{i,j}(t)+\phi_{i,j}(t-1,t)]\} \quad (14)$$

There exist many equivalent paths that only differs in T_i for each component r_i, which translates to many equivalent solutions in neighborhood $N_i(r^{[\tau-1]})$. A simple way to eliminate the redundant equivalent solutions in $N_i(r^{[\tau-1]})$ 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 $T_{-i} = max_{1\leq j\leq n, j\neq i}(T_j)$ at each iteration τ. Then the single agent path planning problem is to find a path r_i with the lowest $F_{p,i}(r;\mu,c,r^{(k-1)})$, 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, $H(n) = lb_i - t(n)$, where $t(n)$ denotes the current time step of the node n. After lb_i, $H(n)$ is set to the Manhattan distance from the current node n to the goal node. Tie breaking can be done by selecting the node with the lowest $t(n)$.

To address the problem of being trapped in unfeasible or inferior locally optimal MAPP solution in the basic local search approach, a stochastic mechanism is incorporated such that the solutions in each local neighborhood is generated probabilistically. To encourage the generation of the optimal solution, the better solution r in $N_i(r^{cur})$ should ideally be assigned a higher generation probability. That is to say, the theoretical generation probability of each solution r in $N_i(r^{cur})$ would be monotonically decreasing with the increasing value of $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14). As a result, given an initial solution, which is the iterated solution $r^{(k-1)}$ in Figure 1, the stochastic local search algorithm iteratively moves to a neighbor solution generated by a local probabilistic model $P(r|r^{cur})$ that is monotonically decreasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$. Since it is only required to find a feasible and improving MAPP solution that is superior to $r^{(k-1)}$ each time, the stochastic local search algorithm can terminate when a feasible solution that satisfies the condition (11) is obtained. The proposed algorithm is called the Probabilistic Iterative Path Coordination (PIPC) algorithm.

Assume $R_i(r^{cur})$ includes all possible path assignments for r_i in $N_i(r^{cur})$, then the local probabilistic model is given by

$$P(r|r^{cur}) = P(r_i|r^{cur}_{-i}) \text{ for each solution } r \in N_i(r^{cur}) \text{ and } r_i \in R_i(r^{cur})(i \in \{1, ..., n\}) \tag{15}$$

4.1.1. 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 $(r_i(0), r_i(1), ..., r_i(T_i))$ 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 $r^{cur}_{-i} = (r^{cur}_1, r^{cur}_2, ..., r^{cur}_{i-1}, r^{cur}_{i+1}, ..., r^{cur}_n)$, we could define the success probability $P_s(r_i|r^{cur}_{-i})$ for each path r_i with respect to the other paths r^{cur}_{-i} as

$$P_s(r_i|r^{cur}_{-i}) = P_s(r_i(0)|r^{cur}_{-i}) \prod_{t=1}^{T_i} P_s(r_i(t)|r_i(t-1), r^{cur}_{-i}) \tag{16}$$

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

$$P_s(r_i(t)|r_i(t) = v) = \eta_i, \text{ where } \eta_i \in [0, 1] \tag{17}$$

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. Assuming a collision node $r_i(t)$ violates k dynamic constraints in (5) and the probability of violating each dynamic constraint is equal to β_i, then the relaxation probability of the collision node $r_i(t) = \bar{v}$ is given by

$$P_r(r_i(t)|r_i(t) = \bar{v}) = \beta_i^k, \text{ where } \beta_i \in [0, 1] \tag{18}$$

The success probability $P_s(r_i(t)|r_i(t) = \bar{v})$ of the collision node $r_i(t) = \bar{v}$ is then given by

$$P_s(r_i(t)|r_i(t) = \bar{v}) = \eta_i * \beta_i^k, \text{ where } \eta_i \in [0, 1] \text{ and } \beta_i \in [0, 1] \tag{19}$$

Next we will define 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. If the path node $r_i(t)$ where $t = 1, 2, ..., T_i$ is a free node v, then the success probability

$$P_s(r_i(t)|r_i(t-1), r_i(t) = v, r^{cur}_{-i}) = P_s(r_i(t)|r_i(t) = v) \tag{20}$$

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

$$P_s(r_i(t)|r_i(t-1), r_i(t) = \bar{v}, r^{cur}_{-i}) = P_s(r_i(t)|r_i(t) = \bar{v}) \tag{21}$$

Since the path node $r_i(0)$ denotes the starting position of the agent A_i and therefore is 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 $r_{-i}^{cur} = (r_1^{cur}, r_2^{cur}, ..., r_{i-1}^{cur}, r_{i+1}^{cur}, ..., r_n^{cur})$ is given by

$$P_s(r_i|r_{-i}^{cur}) = P_s(r_i(0)|r_{-i}^{cur}) \prod_{t=1}^{T_i} P_s(r_i(t)|r_i(t-1), r_{-i}^{cur}) = \eta_i^{f_i(r)} \beta_i^{\Gamma_i(r)} \tag{22}$$

where $r_i \in R_i(r^{cur})$, $\eta_i \in [0,1]$, $\beta_i \in [0,1]$, $\Gamma_i(r) = \sum_{j=1,j\neq i}^{n} \sum_{t=1}^{T_i} \{\Omega_{i,j}(t) + \phi_{i,j}(t-1,t)\} (i \in \{1,2,...,n\})$.

4.1.2. Local Probabilistic Model

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 (22). However, since the probability assignment $P(r_i|r_{-i}^{cur})$ 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, it is difficult to apply techniques like rejection sampling or slice sampling. A naive idea is to generate each possible path r_i in $R_i(r^{cur})$ according to $P(r_i|r_{-i}^{cur})$ independently and then select the best one. Unfortunately, this is computationally expensive. We could however reduce the computational cost by generating the possible paths in $R_i(r^{cur})$ with $P(r_i|r_{-i}^{cur})$ in an ascending order based on the value $F_{p,i}(r;\mu,c,r^{(k-1)})$ in (14). 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

$$P_b(x) = \begin{cases} p & \text{if } x{=}1 \\ 1-p & \text{if } x{=}0 \end{cases} \tag{23}$$

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 mth trial is the first success is given by

$$P_g(r_i^{(m)}|r_{-i}^{cur}) = P_s(r_i^{(m)}|r_{-i}^{cur}) \prod_{i=1}^{m-1} (1 - P_s(r_i^{(i)}|r_{-i}^{cur})), \text{ where } m = 1,2,3,... \tag{24}$$

Unfortunately, the value of $F_{p,i}(r;\mu,c,r^{(k-1)})$ 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 order of $F_{p,i}(r;\mu,c,r^{(k-1)})$. In this way, the possible path in $R_i(r^{cur})$ with lower $F_{p,i}(r;\mu,c,r^{(k-1)})$ can be generated first before others with higher $F_{p,i}(r;\mu,c,r^{(k-1)})$. 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 $R_i(r^{cur})$. The pseudo code of the probabilistic path planner is described in Algorithm 1. 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 $t_v > lb$. 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 10-11), 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. Since node v' is a one-time step successor of the node v, so $c(v,v') = 1 + \mu * \sum_{j=1,j\neq i}^{n} \{\Omega_{i,j}(t_{v'}) + \phi_{i,j}(t_v, t_{v'})\}$. Then the cost $f(v')$ is the sum of $g(v')$ and a time-dependent heuristic function $h(v')$. In our implementation, the heuristic function $h(v') = lb_i - t_{v'}$ if $t_{v'} \leq lb_i$, and

Algorithm 1 ProbabilisticPathPlanner($start$, $goal$, lb_i)

$U \leftarrow \emptyset, Q \leftarrow \emptyset$
$Q \leftarrow Q \cup start$
while $Q \neq \emptyset$ **do**
  $v \leftarrow argmin_{v \in Q} f(v)$
  $Q \leftarrow Q \setminus v$
  $U \leftarrow U \cup v$
  if $v = goal$ and $t_v > lb_i$ **then**
    return true
  end if
  for all $a \in \{N, W, E, S, I\}$ **do**
    $v' \leftarrow successor(v, a)$
    if $v' \notin U$ **then**
      if $v' \notin Q$ and $success(v')$ **then**
        $QQ \cup v'$
        $parent(v') \leftarrow v$
        $g(v') \leftarrow g(v) + c(v, v')$
        $f(v') \leftarrow g(v') + h(v')$
      else if $v' \in Q$ and $g(v') > g(v) + c(v, v')$ and $success(v')$ **then**
        $parent(v') \leftarrow v$
        $g(v') \leftarrow g(v) + c(v, v')$
        $f(v') \leftarrow g(v') + h(v')$
      end if
    end if
  end for
end while
return false

$h(v') = |x_v - x_{goal}| + |y_v - y_{goal}|$ if $t_{v'} > lb_i$. Tie breaking can be done by selecting the node with the lowest time cost $g(v)$. The algorithm returns false if Q is empty.

Thus, the local probabilistic model

$$P(r|r^{cur}) = P(r_i|r^{cur}_{-i}) = 1/Z * P_g(r_i|r^{cur}_{-i}) \text{ for each solution } r \in N_i(r^{cur}), \tag{25}$$

where Z is a normalized constant, $Z = \sum_{r_i \in R_i(r^{cur})} P_g(r_i|r^{cur}_{-i})$.

4.1.3. Parameter tuning strategies

To encourage the convergence of PIPC to a locally optimal solution, the probability assignment $P(r|r^{cur})$ for each solution r in $N_i(r^{cur})$ should ideally decrease with the value of $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14). As such, the performance of the PIPC algorithm is highly dependent on the parameters within $P(r|r^{cur})$ and $F_{p,i}(r; \mu, c, r^{(k-1)})$, namely the control parameter c, the penalty coefficient μ, the minimization parameter η and the relaxation parameter β. Since an incremental scheme for c has been suggested at the beginning of section 4, this section will introduce different parameter setting strategies for the other three parameters.

Condition 1. *$\forall i = 1, 2, ..., n$, $\eta_i = \eta^{w_i}$ and $\beta_i \leq \eta^{\mu}$ where $w_i = 1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])$, $\eta \in (0, 1)$ and $\beta_i \in (0, 1)$, $\mu = M$, where M is a sufficiently high value such that $M > f_i(r)$ for each solution $r \in N_i(r^{cur})$.*

Theorem 1. *If condition 1 holds, $P(r|r^{cur})$ in (25) for $r \in N_i(r^{cur})$ is strictly decreasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14). (See proof of Theorem 1 in appendix)*

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. *$\forall i = 1, 2, ..., n$, $\eta_i = \eta^{w_i}$ and $\beta_i \leq \eta^{\mu}$ where $w_i = 1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])$, $\eta \in (0, 1]$ and $\beta_i \in [0, 1]$, $\mu = M$, where M is a sufficiently high value such that $M > f_i(r)$ for each solution $r \in N_i(r^{cur})$.*

Theorem 2. *If condition 2 holds, $P(r|r^{cur})$ in (25) for $r \in N_i(r^{cur})$ is non-increasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14). (See proof of Theorem 2 in appendix)*

Under condition 2, the monotonic relationship between $P(r|r^{cur})$ and $F_{p,i}(r; \mu, c, r^{(k-1)})$ will still hold even though it is not strictly decreasing like that under condition 1 (See proof of Theorem 2 in appendix).

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. We proposed to achieve this using adaptive parameter setting strategies for μ, β and η.

Adaptive μ strategy - Since the penalty coefficient μ scales the penalty term in $F_{p,i}(r; \mu, c, r^{(k-1)})$, a high μ may cause the penalty term to dominate the cost function while a low μ may slow down the search for a feasible solution. A compromise approach is to set an intermediate value for μ such that the objective function $f_i(r)$ and penalty term at each iteration can be equally considered. This is achieved by updating μ according to the following heuristic:

$$\mu = \sum_{i=1}^{n} \{[1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])] * f_i(r)\} / \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{t=1}^{T_{i,j}} \{0.5 * [\Omega_{i,j}(t) + \phi_{i,j}(t-1, t)]\} \tag{26}$$

The parameter μ can be updated every n iterations, where n is the agent count. When a feasible solution is encountered, which means the denominator in function (26) is 0, we simply set μ to $a * \mu$, where a is a scaling factor between 0 and 1(See Section 5.1.3). Since it is desirable that μ is set high enough in condition 2, a lower bound μ_{min} is imposed. A suitable setting was empirically determined to be set to $1/n$, where n is the agent count.

Adaptive β strategy - The relaxation parameter β only influences the success probability of unfeasible 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 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 unfeasible 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 $F_{p,i}(r;\mu,c,r^{(k-1)})$ in $N_i(r^{cur})$ and will compromise the ability of the algorithm in escaping local optima, especially feasible local optima, 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 trapped around feasible local optima. 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 k the success probability of a solution r associated with η is $\eta^{w_i * f_i(r)}$ ($i \in \{1,...,n\}$). In order to avoid too many rejections, $\eta^{w_i * f_i(r)}$ of the current solution $r^{[\tau-1]}$ should not be too small. As such, a constraint is imposed for η and is given by

$$\eta^{w_i * f_i(r^{[\tau-1]})} \geq \varepsilon_1 \tag{27}$$

where $\varepsilon_1 \in [0,1]$. The function (27) implies $\eta \geq \sqrt[w_i * f_i(r^{[\tau-1]})]{\varepsilon_1}$. Similarly, a constraint is imposed on β_{max} such that β_i is not forced to be a small value. As $\beta_{max} = \eta^{\mu}$, so

$$\eta^{\mu} \geq \varepsilon_2 \tag{28}$$

where $\varepsilon_2 \in [0,1]$. So $\eta \geq \sqrt[\mu]{\varepsilon_2}$. Thus, $\eta_{min} = \max(\sqrt[w_i * f_i(r^{[\tau-1]})]{\varepsilon_1}, \sqrt[\mu]{\varepsilon_2})$. In our implementation, values of $\varepsilon_1 = 0.1$ and $\varepsilon_2 = 0.1$ were found to be suitable.

Given an initial solution $r^{[0]}$, which is set to the iterated solution $r^{(k-1)}$, the PIPC algorithm iteratively generates the next solution by the probabilistic path planner given in Algorithm 1 according to a local probability model $P(r|r^{cur})$ that satisfies the condition 2. The PIPC algorithm terminates when a feasible solution that satisfies the condition (11) is obtained. The pseudo-code of the PIPC algorithm for makespan minimized MAPP is described in Algorithm 2. The function $SolutionGeneration(P(r|r^{[\tau-1]}), N_i(r^{[\tau-1]}))$ tries to generate a new solution from $N_i(r^{[\tau-1]})$ according to the local probabilistic model $P(r|r^{[\tau-1]})$. The scheme *pcUpdate* and *betaUpdate* can either use the fixed parameter setting strategy that sets a fixed value for μ and β, or the proposed adaptive parameter setting strategy for μ and β respectively.

5. Evaluation

The proposed Probabilistic Iterative Makespan Minimization algorithm for makespan minimized MAPP is evaluated in a simulation environment consisting of a 30x20 grid map. Each unit agent is assigned different starting and target positions. The unit time interval is set to dt, which is the time taken by the agent to traverse one grid node.

All the experimental results presented were obtained with the same 100 random initial scenarios. 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 PIMM algorithm, three performance criteria were adopted.

1. *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 the other measures are irrelevant if all n agents are unable to reach their respective goals.
2. *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 criterion 1. This is the optimization objective set out in this work and will determine how fast the proposed algorithm is able to get the required task completed.

Algorithm 2 PIPC

$r^{[0]} \leftarrow r^{(k-1)}$
$i \leftarrow 1, \tau \leftarrow 1, \eta \leftarrow 1$
$\mu \leftarrow pcUpdate(r^{[0]})$
repeat
  repeat
    $\beta \leftarrow betaUpdate(\eta, \mu)$
    $r' \leftarrow SolutionGeneration(P(r|r^{[\tau-1]}), N_i(r^{[\tau-1]}))$
  until a solution r' is successfully generated
  $r^{[\tau]} \leftarrow r'$
  if $r^{[\tau]}$ is feasible **then**
    if $r^{(best)} = \emptyset$ or $F_0(r^{(best)}) > F_0(r^{[\tau]})$ **then**
      $r^{(best)} \leftarrow r^{[\tau]}$
    end if
    $\eta \leftarrow \eta_{min}$
  else
    $\eta \leftarrow 1$
  end if
  $\tau \leftarrow \tau + 1$
  if $i < n$ **then**
    $i \leftarrow i + 1$
  else
    $i \leftarrow 1$ and $\mu \leftarrow pcUpdate(r^{[\tau-1]})$
  end if
until Some stopping criteria are satisfied
return $r^{[\tau-1]}$

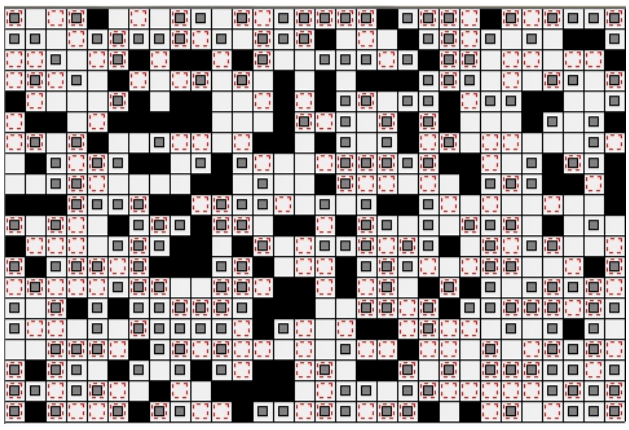

Figure 2. A random scenario in a 30x20 grid map with 200 agents and the static obstacles coverage of 20%. The solid gray squares denote the mobile agents and the dashed squares denote their respective unique destinations.

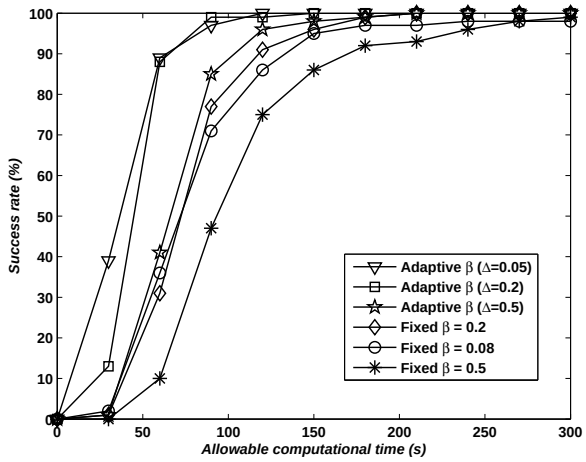

Figure 3. Success rate of PIMM with different β parameter setting strategies against the allowable computational time.

3. *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 criterion 1. This criterion provides a good empirical measure of the time complexity of the proposed algorithm.

5.1. Evaluation of the Probabilistic Iterative Makespan Minimization Algorithm

The performance of PIMM is dependent on the chosen values of the relaxation parameter β, minimization parameter η, the penalty coefficient μ and the control parameter c. A simple incremental scheme has been recommended for c. By contrast, two main strategies for setting the values of η, β and μ have been suggested in section 4.1.3, namely the fixed and adaptive parameter setting strategies. The following experiments firstly evaluated if the adaptive parameter setting strategies of η, β and μ can yield superior performances over fixed parameter setting strategies. We then compared the performance of PIMM under different incremental rates ε_c for control parameter c.

Since PIMM is an iterative algorithm that can improve the solution quality with increasing computational cost, its progressive performance is plotted against the allowable computational time. The performance of this algorithm was evaluated and compared on a 30x20 grid map with an agent count of 200 and static obstacle coverage of 20%, as shown in Figure 2.

5.1.1. Evaluation of PIMM with different β parameter setting strategies

The following experimental results evaluate the fixed and adaptive β strategies in influencing the performance of PIMM. Three different fixed values of $\beta = 0.08$, $\beta = 0.2$ and $\beta = 0.5$ were chosen for the fixed β strategy while the adaptive β strategy gradually increases the value of β from 0 to η^{μ} by an interval $\Delta(0 \leq \Delta \leq \eta^{\mu})$ until a new solution r' is successfully obtained at each iteration of PIPC. As such, three different incremental rates Δ for adaptive β, namely $\Delta = 0.05$, 0.2 and 0.5 were also evaluated. The comparisons were done with the other parameters set to $\mu = 100$, $\eta = 1$ and incremental rate $\varepsilon_c = 0.01$ for control parameter c.

A high β increases the probability that unfeasible 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, a high β (ie. $\beta = 0.5$) will lead to a correspondingly slower increase in success rate compared to a low β (ie. $\beta = 0.2$) (see Figure 3). However, if β is too low (ie. $\beta = 0.08$), the probability of rejections will increase rapidly leading to a higher computational cost at each iteration of PIPC. This will also reduce the success rate performance for a given allowable computation cost. In short, there is an optimal choice for β that will give the best success rate performance and this optimal choice was observed to be scenario dependent and therefore difficult to be determined in practice. The adaptive β strategy attempts to find the ideal value of β at each iteration of PIPC by adopting a systematic procedure that progressively increase the value of β. This adaptive β strategy was observed

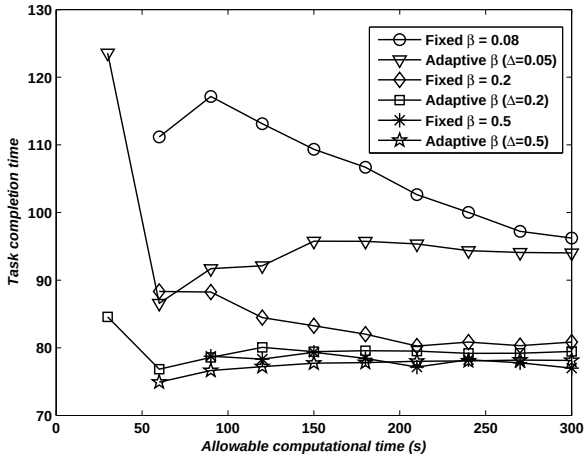

Figure 4. Task completion time of PIMM with different β parameter setting strategies against the allowable computational time.

to have a faster increase in success rate with allowable computational time than any of the fixed β values used (see Figure 3).

As for the task completion time performance, 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, conversely, 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 than a low value of β. Having a better success rate performance, the adaptive β strategy is also able to produce a superior task completion time performance that is comparable with the fixed $\beta = 0.5$ value (see Figure 4).

The chosen value of the interval Δ used in the adaptive β strategy also influences PIMM'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 (e.g. $\Delta = 0.05$) 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 3). On the other hand, a larger interval Δ(e.g. $\Delta = 0.5$) often leads to a higher over-estimation of β, which actually results in a superior task completion time performance, like the case observed with a high fixed β value of 0.5 (see Figure 4). The best compromise is the intermediate value of $\Delta = 0.2$, which is able to achieve the good success rate performance of $\Delta = 0.05$ and the superior task completion time performance of $\Delta = 0.5$.

5.1.2. Evaluation of PIMM with different η parameter setting strategies

Adopting the adaptive β strategy presented in the previous section, we next investigated the performance improvement if an adaptive η strategy was also adopted. 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 switches its value between 1.0 and η_{min}. The penalty coefficient μ is set to a fixed value of 100 and the incremental rate ε_c is set to 0.01.

Under a high value of $\mu = 100$, the constraints (27) and (28) prevent the value of η_{min} from being set too low and therefore ameliorate the problem of rejections at each iteration of PIPC. As a result, different fixed values of η cause a trivial difference on the success rate performance of PIMM for a given allowable computational cost (see Figure 5). However, in terms of task completion time, PIMM with a low η value is more likely to find a superior makespan-minimized MAPP solution (see Figure 6) for a given high enough computational time. 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 local optima. 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 (see Figure 5 and 6).

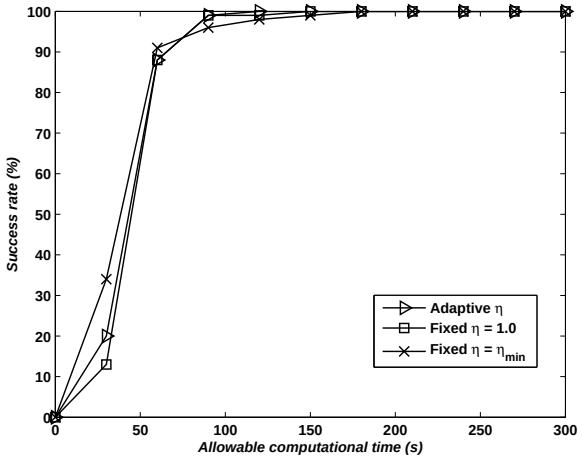

Figure 5. Success rate of PIMM with different η parameter setting strategies against the allowable computational time.

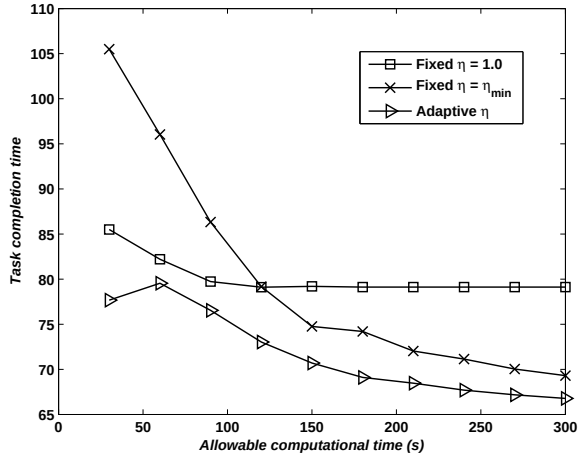

Figure 6. Task completion time of PIMM with different η parameter setting strategies against the allowable computational time.

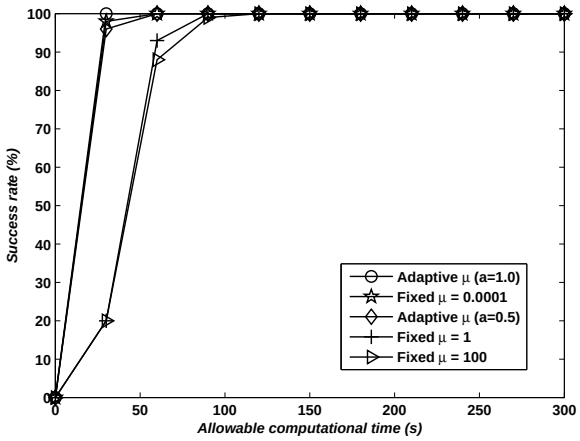

Figure 7. Success rate of PIMM with different μ parameter setting strategies against the allowable computational time.

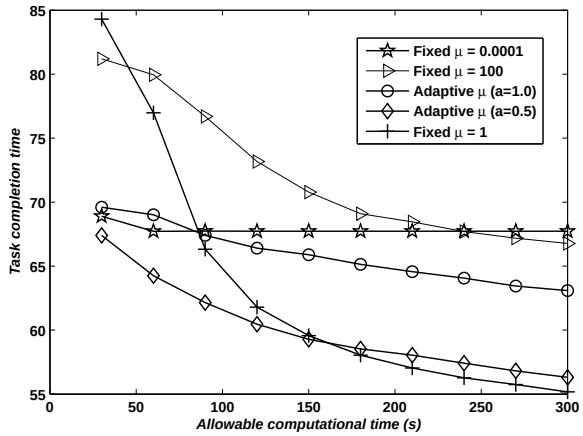

Figure 8. Task completion time of PIMM with different μ parameter setting strategies against the allowable computational time.

5.1.3. Evaluation of PIMM with different μ parameter setting strategies

The adaptive β and η strategies were observed to improve PIMM's performance under a fixed value of $\mu = 100$. The next set of experiments investigates if further performance improvements can be obtained if an adaptive μ strategy was also adopted. Experiments were done comparing the adaptive μ strategy using the update scheme suggested in (26) against three fixed values of $\mu = 0.0001$, $\mu = 1$ and $\mu = 100$. Two different scaling factors $a = 1.0$ and $a = 0.5$ for adaptive μ were used. All the comparisons were done under the incremental rate $\varepsilon_c = 0.01$.

Since the penalty coefficient μ scales the penalty term in the cost function (13), so varying the value of μ changes the selection priorities between feasible solutions and unfeasible solutions. Though a low value of μ favors the selection of unfeasible solutions, a biased search towards feasible solutions in adaptive β still leads to a low selection probability of unfeasible solutions. As a result, a high value of μ does not produce a faster increase of success rate with allowable computational time than a low value of μ. However, a high value of μ will incur a higher computational cost at each iteration of PIPC, as it leads to more free nodes being explored. In this case, for a given computational cost($< 100s$), a lower value of μ may result in a higher success rate(see Figure 7). Moreover, a lower value of μ allows a smaller η_{min} and therefore μ=1 produces a superior makespan minimized MAPP solution compared to $\mu = 100$. However, setting μ too low (eg. μ=0.0001) fails to satisfy the condition 2 and therefore deteriorate the optimal solution convergence of PIMM. Thus, finding an optimal value of μ is a challenging task. By contrast, the

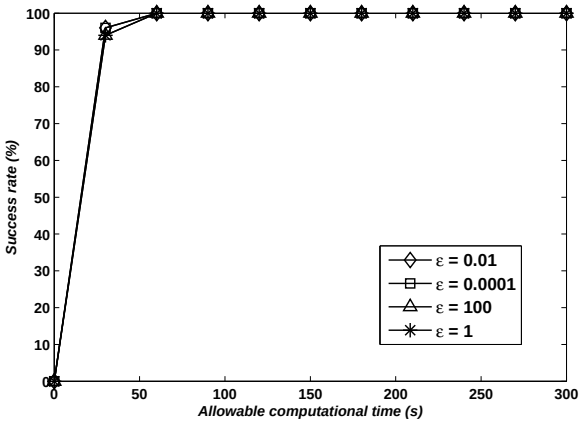

Figure 9. Success rate of PIMM with different ε_c parameter setting strategies against the allowable computational time.

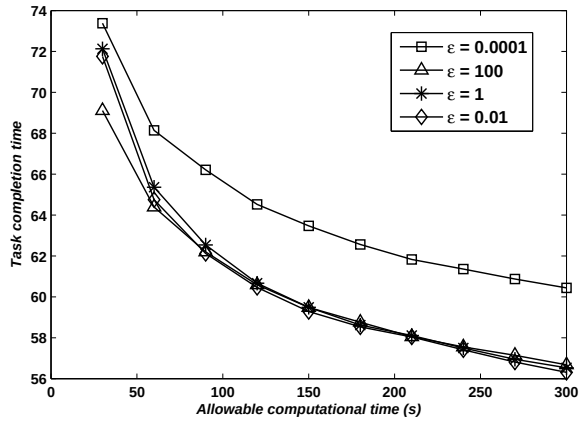

Figure 10. Task completion time of PIMM with different ε_c parameter setting strategies against the allowable computational time.

adaptive μ strategy dynamically set its value such that the influence of the time objective function and the penalty term remains relatively balanced when searching for a feasible solution, and gradually decreases($a < 1.0$) when being trapped around a feasible local optimum. In this way, it was observed that the adaptive μ strategy with $a < 1.0$ in PIMM leads to a superior performance in terms of both success rate and task completion time compared to the fixed μ strategy.

In summary, fixed value strategies for η, β and μ suffers from a tradeoff between intensive and extensive search in the MAPP solution space. By contrast, the proposed adaptive parameter setting strategies allow the values of η, β and μ to be dynamically changed by some heuristics that can determine when it is most appropriate to adopt either an extensive or intensive search. It was shown that this informed search in the MAPP solution space led to improved success rate and task completion time performance.

5.1.4. Evaluation of PIMM with different incremental rates ε_c for control parameter c

The following experimental results evaluate different incremental rates ε_c in influencing the performance of PIMM. Four different incremental rate values of $\varepsilon_c = 0.0001$, $\varepsilon_c = 0.01$, $\varepsilon_c = 1$ and $\varepsilon_c = 100$ were used.

The control parameter c is only designed to navigate the search for a makespan minimized MAPP solution, as a result, it has negligible influence on the success rate performance of PIMM (See Figure 9). Since the approximate problem in (8) is equivalent to the makespan minimized MAPP problem when c is sufficiently high, a low incremental

rate ε_c leads to a slow local convergence to a makespan minimized MAPP solution compared to a higher value. However, the better local convergence associated with a high ε_c compromises the extensive search in the MAPP solution space. A low incremental rate ε_c (eg. $\varepsilon_c = 0.01$) is more likely to find a superior makespan minimized MAPP solution if more computational cost is permitted.

5.2. Comparative Performance Evaluation of the PIPC Algorithm

The main objective of this work is to find a makespan-minimized solution that has a short task completion time. Since not many MAPP algorithms were specifically designed with this objective in mind, we compared the PIMM's success rate, computational time and task completion time performances with other MAPP algorithms that were designed with other performance criteria in mind. The windowed hierarchical cooperative A* (WHCA*) algorithm [15] is computationally efficient to find a good makespan MAPP solution. Comparison was done with WHCA* using different window steps. The second MAPP algorithm is the Optimal Anytime (OA) algorithm [29], which iteratively refines the solution with respect to a time related cumulative cost function until some stopping criteria are satisfied. This algorithm is an iterative and optimal MAPP algorithm but is not designed for the makespan-minimized MAPP problem. The third one is the Guided Iterative Prioritized Planning (GIPP) algorithm [30], which is closest in concept to PIMM as it iteratively improves the solution within a sequence of local neighborhoods until some stopping criteria are satisfied. In other words, the quality of the solution can also be improved by increasing the computational cost. However, unlike the stochastic PIMM algorithm, GIPP is a deterministic algorithm. The final one is the Yu and LaValle's Integer Linear Programming (ILP) based algorithm [37]. Like the PIMM algorithm, ILP was also specifically designed to solve the makespan-minimized MAPP problem. All benchmark algorithms used were re-implemented based on published works except ILP [37]. The GIPP [30] algorithm was originally developed by the authors and the ILP [37] algorithm was modified from the source code obtained from [40].

The performance of these MAPP algorithms was compared on a 30x20 grid map with a static obstacle density of 20% (see Figure 2), over an agent count ranging from 0 to 240. Another series of evaluations was done using 120 agents with the static obstacle density varying from 0% to 40%. For each run with a different agent count or static obstacle density, 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.

5.2.1. Comparative evaluation under varying agent count

The following experimental results compare the performances of ILP, the WHCA* algorithms with 16 (WHCA*(16)) and 128 window steps (WHCA*(128)), OA, GIPP and PIMM in a 30x20 grid map with static obstacle density of 20% over varying agent count. A high value of $\mu = 100$ is used in GIPP. The adaptive strategies, namely adaptive β with incremental rate $\Delta = 0.2$, adaptive η and adaptive μ with scaling factor $a = 0.5$, and the incremental rate $\varepsilon_c = 0.01$ for control parameter c are used in PIMM.

OA has an exponentially increasing state search space with the agent count and the depth of the goal state for each group. While for ILP, the increasing agent count will significantly increase its number of flow edges and therefore leads to a rapidly increasing search space. As a result, despite the already high timeout period of 5 minutes, the success rate of the optimal algorithms OA and ILP still drops rapidly as the agent count increases (see Figure 11). By contrast, the computationally efficient MAPP algorithm WHCA* did find a good solution more quickly (see Figure 13), but still suffers from poor success rate. Though a longer window step improves its success rate performance, it comes at the expense of a higher computational time and the inferior solution quality. GIPP was able to achieve superior performances over WHCA* in all three performance criteria. However, the greedy way of accepting improving solutions makes it more difficult for GIPP to escape local optima. Moreover, some of the local optima are unfeasible solutions, which affect GIPP's success rate performance. As a result, GIPP's success rate performance is inferior to PIMM's (see Figure 11). For a given computational time, it was also observed that PIMM was more likely to produce a superior makespan minimized solution than GIPP, especially in crowded environments. ILP, being an optimal algorithm, provides a good indication of the optimal task completion time. However, compared with ILP's high computational demand, the proposed PIMM algorithm strikes a good tradeoff between the quality of the makespan minimized solution and the computational cost.

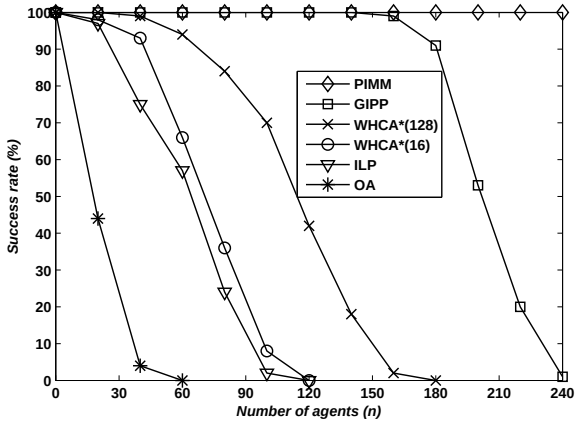

Figure 11. Comparative success rate under varying agent count.

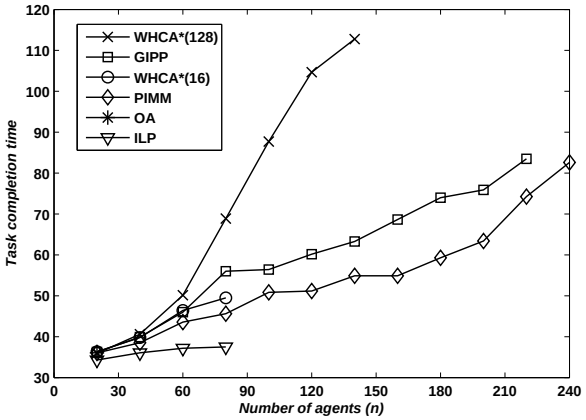

Figure 12. Comparative task completion time under varying agent count.

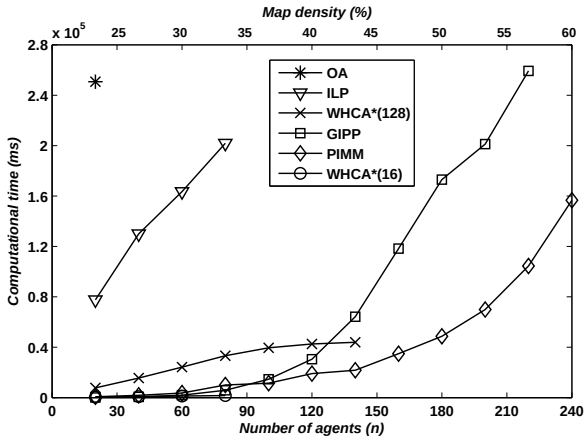

Figure 13. Comparative computational time under varying agent count.

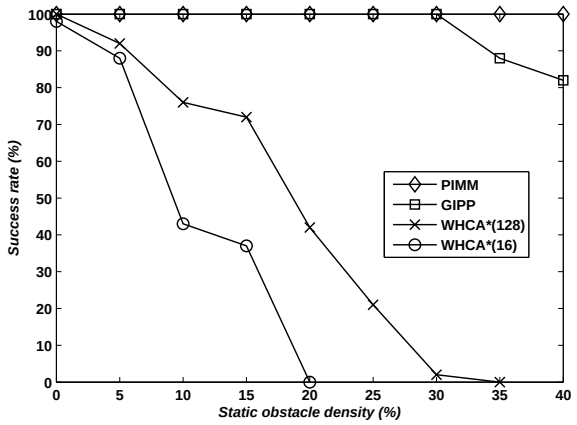

Figure 14. Comparative success rate under varying static obstacle density.

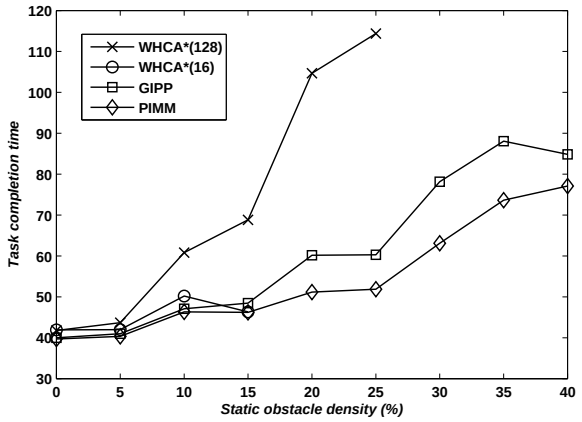

Figure 15. Comparative task completion time under varying static obstacle density.

5.2.2. Comparative evaluation under varying static obstacle density

The next set of experiment evaluates the performance of the PIMM algorithm when the map density (i.e. agent density + static obstacle density) is increased using increasing static obstacles instead of agent count. Like before, comparative evaluation was done between ILP, the WHCA* algorithms with 16 window step (WHCA*(16)) and 128 window step (WHCA*(128)), OA, GIPP and PIMM in a 30x20 grid map under varying static obstacle density with a fixed agent count of 120. Parameter settings for GIPP and PIMM were similar to those used in section 5.2.1.

Due to the insufficient timeout of 5 minutes, OA and ILP both failed in the environment under varying static obstacle density with an agent count of 120. As such, they were not featured in the results presented in Figures 14 to 16. PIMM continues to exhibit the best success rate and task completion time performance amongst all the other MAPP algorithms. It was observed that, for the same given map density, increasing static obstacle density does not increase the computational cost of PIMM as rapidly as increasing agent count (compare Figures 13 and 16, comparable map density is indicated on the top of both plots). This is because increasing static obstacle density does not increase the size of the solution space unlike increasing agent count. In addition, solutions violating the static constraints introduced by static obstacle are strictly satisfied to avoid unnecessary search. This characteristic of PIMM is particularly advantageous as most practical indoor MAPP applications such as the planning of autonomous transportation vehicles in a factory floor have a relatively small number of mobile agents but a high static obstacle density (e.g. corridors, narrow passage way, etc).

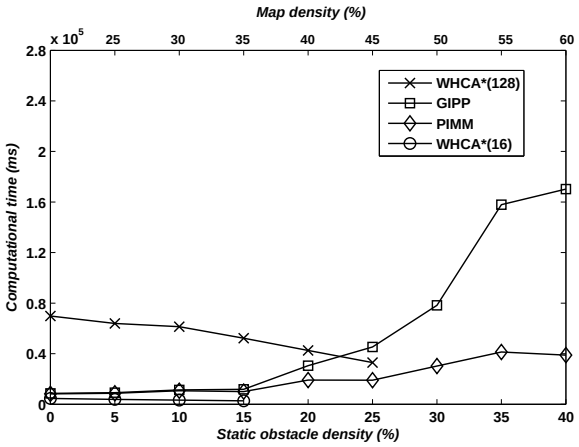

Figure 16. Comparative computational time under varying static obstacle density.

6. Conclusion

Finding an optimal solution for makespan minimized MAPP in a grid map can be intractable since the size of the search space increases rapidly with the number of agents and the map size. This paper proposed a stochastic algorithm called Probabilistic Iterative Makespan Minimization that tries to approximate the optimal solution of makespan minimized MAPP by solving a sequence of computationally simpler MAPP sub-problems. At each iteration, a novel local search algorithm is used to find a sufficiently good solution. In order to address the problem of being trapped in local optima, the proposed Probabilistic Iterative Path Coordination algorithm adopts a stochastic mechanism that accepts each neighboring solution based on a local probabilistic model. This model assigns the success probability for each neighboring solution based on a target cost function that measures the quality of the makespan minimized MAPP solution. A minimization parameter η and a relaxation parameter β are introduced to model the success probability of each solution. Unfortunately, the performance of the PIMM algorithm is highly sensitive to several inherent parameters. In order to avoid the need to constantly fine tune these parameters for different scenarios, adaptive parameter setting strategies were employed to automatically adjust the values of these parameters based on current and historical information obtained during the search process. This adaptive parameter tuning strategy also allowed the incorporation of heuristics that can dynamically switch between an extensive or intensive solution search strategies based on the current state of the local search. This significantly enhances the efficiency of the iterative process of searching for a solution with improving quality. The experimental results from comparative studies show that the proposed PIMM algorithm with adaptive parameter setting strategy was indeed able to outperform all the other MAPP algorithms in finding good quality makespan minimized solutions using the least amount of computational time, especially in crowded environments.

References

[1] J.L. Zhou and A.L. Tits, Nonmonotone Line Search for Minimax Problems, Journal of optimization theory and applications. Vol.76,no.3,(1993).
[2] Z.B. Zhou, X. Cai, J. Jian, An Improved SQP algorithm for solving minimax problems, Applied Mathematics Letters 22, 464-469, (2009).
[3] 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, (2008),pp.581-589.
[4] F.S. Wang, K.C. Zhang, A hybrid algorithm for minimax optimization, Annals of Operations Research 164, (2008), pp.67-191.
[5] F. Wang and Y. Wang, Nonmonotone algorithm for minimax optimization problems. Applied Mathematics and Computation, (2011).
[6] J. Li and J. Huo, Inexact smoothing method for large scale minimax optimization. Journal of Applied Mathematics and Computation, vol. 218, (2011), pp. 2750-2760.
[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-Robot Path Planning is Intractable, Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.

[9] P. Svestka and M. H. Overmars, Coordinated path planning for multiple robots, Robotics and Autonomous Systems. Vol.23,no.3,(1998),125–152.
[10] G. Sharon, R. Stern, M. Goldenberg, A. Felner, The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding, in proceedings of the twenty-second International Joint Conference on Artificial Intelligence. Vol.1,(2011), 662-667.
[11] 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).
[12] A.t. Mors, C. Witteveen, J. Zutt and F. A. Kuipers, Context-Aware Route Planning, 8th German Conference, MATES 2010, Leipzig, Germany. (2010),pp.138-149.
[13] 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. (2011),pp.1091-1092.
[14] 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,(2001),pp.271-276.
[15] D. Silver, Cooperative pathfinding, in AIIDE. (2005),pp.117-122.
[16] N. Sturtevant and M. Buro. Improving Collaborative Pathfinding Using Map Abstraction. In Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference, (2006), pp. 80-85.
[17] M. Ryan, Multi-robot path planning with subgraphs, in Proc. of the 19th Australasian Conference Robotics and Automation.(2006).
[18] M. Ryan, Graph Decomposition for Efficient Multi-robot Path Planning, Proceedings of the 20th international joint conference on Artifical intelligence. (2007), pp.2003-2008.
[19] 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).
[20] 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).
[21] E. Masehian and A.H. Nejad, Solvability of Multi Agent Motion Planning Problems on Trees, IEEE International Conference on Intelligent Agents and Systems.(2009).
[22] 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).
[23] 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).
[24] 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.(2011),pp.55-90.
[25] 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. (2011),pp.294-300.
[26] 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).
[27] 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).
[28] T. Standley, Finding Optimal Solutions to Cooperative Pathfinding Problems, in proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. (2010),pp.173-178.
[29] T. Standley and R. Korf, Complete Algorithms for Cooperative Pathfinding Problems, Proceedings of the 22nd International Joint Conference on Artificial Intelligence. (2011),pp.668-673.
[30] W. Wang and W.B. Goh, Time Optimized Multi-Agent Path Planning using Guided Iterative Prioritized Planning, Proceedings of 12th International Conference on Autonomous Agents and Multi-Agent Systems.(2013),pp.1183-1183.
[31] B.d. Wilde, Cooperative Multi-Agent Path Planning. Master Thesis, Delft University of Technology. (2012).
[32] J-P. Watson, An Introduction to Fitness Landscape Analysis and Cost Models for Local Search, in M. Gendreau, J.-Y. Potvin(Ed.2), Handbook of Metaheuristics. (pp. 599-623),Springer.(2010).
[33] D. Ratner, M.K. Warmuth, Finding a Shortest Solution for the N x N Extension of the 15-PUZZLE Is Intractable, National Conference on Artificial Intelligence. (1986).
[34] P. Surynek. A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs. In IEEE International Conference on Robotics and Automation, (2009), pp.3613-3619.
[35] P. Surynek. An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning. In IEEE International Conference on Tools with Artificial Intelligence, (2009), pp. 151-158.
[36] 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, (2011), pp. 3260-3267.
[37] J. Yu and S. M. LaValle. Time Optimal Multi-agent Path Planning on Graphs. The First AAAI Workshop on Multiagent Pathfinding, (2012).
[38] A. Krontiris, R. Luna and K. E. Bekris. From Feasibility Tests to Path Planners for Multi-Agent Pathfinding. Symposium on Combinatorial Search, (2013).
[39] M. B. Bazaraa, H. D. Sherali and C. M. Shetty. Nonlinear programming, theory and algorithms, 3rd, Chapter 10 Methods of Feasible Directions, (2006), pp.537-654.
[40] J. Yu and S. M. LaValle. Planning Optimal Paths for Multiple Robots on Graphs. Retrieved from http://msl.cs.uiuc.edu/ jyu18/_mapp.html, (2013).

Appendix A.

The proofs of Theorem 1 and Theorem 2 are as follows:

Lemma 1. *Assume* $w_i = 1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])]$ *and* μ *is set to a sufficiently high value such that* $\mu > f_i(r)$ *for* $r \in N_i(r^{cur})$*. For any two solutions* $r^{(a)}$ *and* $r^{(b)}$ *in* $N_i(r^{cur})$*,* $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$ *only if* $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$ *or* $f_i(r^{(a)}) < f_i(r^{(b)})$ *given* $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$.

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

If $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$

$$\begin{aligned} F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)}) &= w_i * f_i(r^{(b)}) + \mu * \Gamma_i(r^{(b)}) \geq \{w_i * f_i(r^{(b)})\} + \mu * [\Gamma_i(r^{(a)}) + 1] \\ &> \{w_i * f_i(r^{(b)})\} + \mu * \Gamma_i(r^{(a)}) + f_i(r^{(a)}) \\ &\geq \{w_i * f_i(r^{(b)})\} + \mu * \Gamma_i(r^{(a)}) + \{w_i * f_i(r^{(a)})\} \geq F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) \end{aligned} \tag{A.1}$$

So $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$ always hold if $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$.

If $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$,

$$\begin{aligned} F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)}) &= w_i * f_i(r^{(b)}) + \mu * \Gamma_i(r^{(b)}) \\ &= w_i * f_i(r^{(b)}) + \mu * \Gamma_i(r^{(a)}) \end{aligned} \tag{A.2}$$

So given $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$, $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$ only if $f_i(r^{(a)}) < f_i(r^{(b)})$.

If $\Gamma_i(r^{(a)}) > \Gamma_i(r^{(b)})$, according to (A.1),

$$F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) > F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)}) \tag{A.3}$$

In summary, $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$ only if $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$ or $f_i(r^{(a)}) < f_i(r^{(b)})$ given $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$. □

Lemma 2. *Assume* $\eta_i = \eta^{w_i}$ *and* $\beta_i \leq \eta^{\mu}$ *where* $i = 1, 2, ..., n$, $w_i = 1/\sum_{j=1}^{n} \exp(c * [f_j(r^{(k-1)}) - f_i(r^{(k-1)})])$*,* μ *is set to a sufficiently high value such that* $\mu > f_i(r)$ *for* $r \in N_i(r^{cur})$*, then* $P_s(r_i r_{-i}^{cur})$ *in (25) is strictly decreasing with* $F_{p,i}(r; \mu, c, r^{(k-1)})$ *in (14) when* $\eta \in (0, 1)$ *and* $\beta_i \in (0, 1)$*, and non-increasing with* $F_{p,i}(r; \mu, c, r^{(k-1)})$ *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 $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$, according to lemma 1, $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$ or $f_i(r^{(a)}) < f_i(r^{(b)})$ given $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$.

1. When $\Gamma_i(r^{(a)}) < \Gamma_i(r^{(b)})$:

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

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

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

   $$P_s(r_i | r_{-i}^{cur}) = \eta^{w_i * f_i(r) + (\mu+\Delta) * \Gamma_i(r)} \tag{A.5}$$

   Since $w_i * f_i(r^{(a)}) + \mu * \Gamma_i(r^{(a)}) < w_i * f_i(r^{(b)}) + \mu * \Gamma_i(r^{(b)})$, so

   $$w_i * f_i(r^{(a)}) + (\mu + \Delta) * \Gamma_i(r^{(a)}) < w_i * f_i(r^{(b)}) + (\mu + \Delta) * \Gamma_i(r^{(b)}) \text{ if } \Delta < + \tag{A.6}$$

   While

   $$w_i * f_i(r^{(a)}) + (\mu + \Delta) * \Gamma_i(r^{(a)}) \leq w_i * f_i(r^{(b)}) + (\mu + \Delta) * \Gamma_i(r^{(b)}) \text{ if } \Delta = + \tag{A.7}$$

   Thus

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

   And

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

If $\eta_i = 1$, then $\beta_i \in [0, 1]$,

$$P_s(r_i|r_{-i}^{cur}) = \beta_i^{\Gamma_i(r)} \tag{A.10}$$

So

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

And

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

2. When $\Gamma_i(r^{(a)}) = \Gamma_i(r^{(b)})$ and $f_i(r^{(a)}) < f_i(r^{(b)})$:

If $\eta_i \in (0, 1)$,

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

And

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

If $\eta_i = 1$,

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

In summary, for any two solutions $r^{(a)}$ and $r^{(b)}$ in $N_i(r^{cur})$ such that $F_{p,i}(r^{(a)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(b)}; \mu, c, r^{(k-1)})$, $P_s(r_i^{(a)}|r_{-i}^{cur}) > P_s(r_i^{(b)}|r_{-i}^{cur})$ when $\eta_i \in (0, 1)$ 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 $\eta_i \in (0, 1]$ and $\beta_i \in [0, 1]$. □

Theorem 1. *If condition 1 holds, $P(r|r^{cur})$ in (25) for $r \in N_i(r^{cur})$ is strictly decreasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14).*

Proof. Since the solution r in $N_i(r^{cur})$ with lower $F_{p,i}(r; \mu, c, r^{(k-1)})$ is given the higher priority trial order, then for any two solutions $r^{(m)}$ and $r^{(m+1)}$ $(m = 1, 2, ...)$ in $N_i(r^{cur})$, $F_{p,i}(r^{(m)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(m+1)}; \mu, c, r^{(k-1)})$. If condition 1 holds, according to lemma 2,

$$P_s(r_i^{(m)}|r_{-i}^{cur}) > P_s(r_i^{(m+1)}|r_{-i}^{cur}) \tag{A.16}$$

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

$$\prod_{j=1}^{m-1}(1 - P_s(r_i^{(j)}|r_{-i}^{cur})) > \prod_{j=1}^{m}(1 - P_s(r_i^{(j)}|r_{-i}^{cur})) \tag{A.17}$$

Then

$$P_g(r_i^{(m)}|r_{-i}^{cur}) > P_g(r_i^{(m+1)}|r_{-i}^{cur}) \tag{A.18}$$

As a result, $P_g(r_i|r_{-i}^{cur})$ is strictly decreasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ if condition 1 holds. As $P(r|r^{cur}) \propto P_g(r_i|r_{-i}^{cur})$ and therefore $P(r|r^{cur})$ is strictly decreasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ if condition 1 holds. □

Theorem 2. *If condition 2 holds, $P(r|r^{cur})$ in (25) for $r \in N_i(r^{cur})$ is non-increasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ in (14).*

Proof. Since the solution r in $N_i(r^{cur})$ with lower $F_{p,i}(r; \mu, c, r^{(k-1)})$ is given the higher priority trial order, for any two solutions $r^{(m)}$ and $r^{(m+1)}$ $(m = 1, 2, ...)$ in $N_i(r^{cur})$, $F_{p,i}(r^{(m)}; \mu, c, r^{(k-1)}) < F_{p,i}(r^{(m+1)}; \mu, c, r^{(k-1)})$. If condition 2 holds, according to lemma 2,

$$P_s(r_i^{(m)}|r_{-i}^{cur}) \geq P_s(r_i^{(m+1)}|r_{-i}^{cur}) \tag{A.19}$$

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

$$\prod_{j=1}^{m-1}(1 - P_s(r_i^{(j)}|r_{-i}^{cur})) \geq \prod_{j=1}^{m}(1 - P_s(r_i^{(j)}|r_{-i}^{cur})) \tag{A.20}$$

Then

$$P_g(r_i^{(m)}|r_{-i}^{cur}) \geq P_g(r_i^{(m+1)}|r_{-i}^{cur}) \tag{A.21}$$

As a result, $P_g(r_i|r_{-i}^{cur})$ is non-increasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ if condition 2 holds. As $P(r|r^{cur}) \propto P_g(r_i|r_{-i}^{cur})$ and therefore $P(r|r^{cur})$ is non-increasing with $F_{p,i}(r; \mu, c, r^{(k-1)})$ if condition 2 holds. □