Deadlock Prevention in Rendezvous Generation for On-demand Inter-robot Resource Delivery

Yin Chen, Xinjun Mao and Fu Hou
College of Computer, National University of Defense Technology, Changsha, Hunan, China

Keywords: Deadlock, Rendezvous, Resource, Multi-robot System.

Abstract: In this paper, we consider a multi-robot system (MRS) which executes task points associated with 2-D locations. Each task point demands certain physical resources for its execution. A robot can fetch these resources either from fixed stations, or by conducting rendezvouses with other robots who happen to possess these resources, provided that the latter option can be more beneficial in terms of cost or resource availability. However, applying rendezvouses may cause deadlock among robots, through (1) the tangling among rendezvouses, or (2) sabotaging the resource consistency of the schedule of a robot which is originally holding the resources. We analyse these problems and introduce a series of deadlock prevention rules that are embedded into an A*-based rendezvous planning algorithm, so that both type of deadlocks can be avoided in the rendezvouses.

1 INTRODUCTION

Multi-robot systems (MRS) are applied in many domains where their task execution requires retrieving physical resources from the environment. The required resources may not be available in the vicinity of a robot or may be held by other robots, making it a desirable choice for a robot in urgent need of a particular set of resources to rendezvous with other robots and fetch the resources from these robots, rather than travel to the fixed resource stations which are either far from the robot or depleted. Assume each robot possesses a sequence of task points, each demanding a specific set of resources, which can be fetched from stations or via rendezvous with other robots. However rendezvouses for resource delivery may cause two types of inter-robot deadlocks which may block task execution: (1) **Rendezvous tangling**, i.e., the rendezvouses form a loop waiting for each other to complete, blocking the execution forever. (2) **Demand looping**, i.e., when all robots are self-interested, they may be unable to achieve a globally feasible rendezvous plan that satisfy each robot's own demands without jeopardising others' demands.

In this paper we present deadlock prevention mechanisms for the two types of deadlocks respectively. The first type is tackled by constraining the insertion index of each rendezvous so as to prevent tangling. The second type is tackled by coordinating the rendezvous planning on different robots so that each rendezvous respects the demands of entire MRS rather than only individual robots. We then present a centralised A*-based algorithm (which will be transformed into a distributed version in the future) that searches for the best rendezvous insertion sequence for satisfying the resource demands of entire MRS, while respecting all the deadlock prevention rules.

2 RELATED WORK

Our work deals with the resource demands of schedules of a MRS by dynamically generating rendezvouses for fetching resources. In robotics literature, our work is similar in some way to the multi-robot recharging problem, in which the robots (especially UAVs) rendezvous with ground-based charging robots in order to charge their batteries. (Keshmiri, 2011) presents an opportunistic control approach to address the multi-robot multi-rendezvous recharging problem, which combines centralised and distributed decision-making. (Neil Mathew, 2013) and (Neil Mathew, 2015) present a method for multi-robot recharging problem by modelling the rendezvous planning problem into an integer linear program and transform it into a travelling salesperson problem. Despite their similarity: (1) our work considers heterogeneous demands by different task points for physical resources, and the types of resource are not limited to energy; (2) in stead of absolutely differenti-

Chen Y., Mao X. and Hou F.
Deadlock Prevention in Rendezvous Generation for On-demand Inter-robot Resource Delivery.
DOI: 10.5220/0006225806700675
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 670-675
ISBN: 978-989-758-220-2

ating between resource provider and consumer, we make it possible for each robot to take any of both roles according to the demands, so that the system can more flexibly react to environment changes; (3) the deadlock problem is raised because of the interdependency among different task points in the schedules, which is largely omitted in the traditional recharging problem. For the resource-caused deadlock problem, (Han Lei, 2014) presents a Petri-Net-based multi-robot scheduling approach for flexible manufacturing system, which embeds deadlock control policies into heuristic (e.g., genetic) search algorithm. Our work is partially inspired by their idea of embedding deadlock prevention mechanisms into search algorithm. In comparison, our work assumes dynamic environment and unforeseeable requirement resources, which make our work potentially more applicable to a dynamic environment rather than a relatively fixed industrial environment.

3 PROBLEM FORMATION

3.1 Preliminaries

Task Point. Each *task point* π is a unit of task execution on each robot, and is written as a tuple $\langle \kappa, l, R_{\triangleleft}, R_{-}, R_{+} \rangle$, where (1) κ is the action to be performed by the robot to which the task point π is assigned; (2) $l \in \mathbb{R}^2$ is the 2-D location at which the π is to be executed; (3) $R_{\triangleleft}$ is the set of resources required by π for its successful execution; (4) $R_{-} \subseteq R_{\triangleleft}$ is the set of resources consumed by π; (5) R_{+} is the set of resources generated by π, provided that κ is a resource-fetching action.

Schedule. Task points are organised into schedules in order to be executed on robots. Each robot possesses a *schedule* σ which is defined as a sequence of task points $\pi_1, ..., \pi_n$, where $|\sigma| = n$ is the length of the schedule. For each $i = 1, ..., |\sigma|$, $\sigma(i) = \pi_i$ is the i^{th} task point of the σ, and $\sigma(0)$ is a presumed (but not really existing) task point which marks the very beginning of the schedule. The index of π is $\iota_\sigma(\pi)$. For each task point $\pi \in \sigma$, we make the following definitions: (1) the set of task points preceding π is $\Pi_{\triangleleft}(\pi) = \{\pi' \in \sigma : \iota_\sigma(\pi') < \iota_\sigma(\pi)\}$; (2) the set of task points succeeding π is $\Pi_{\triangleright}(\pi) = \{\pi' \in \sigma : \iota_\sigma(\pi') > \iota_\sigma(\pi)\}$.

Resource. Each resource r is defined as a tuple $\langle y, m \rangle$, where (1) y is the type of r; (2) m is the amount of resource of type y shown by r. A set of resources R can thus be defined as $\{r = \langle y, m \rangle : y \in Y, m \in \mathbb{R}\}$.

Operations of Resources. In this work, we assume the resources are linear, and define their operations: (1) $\langle y, m_1 \rangle + \langle y, m_2 \rangle = \langle y, m_1 + m_2 \rangle$; (2) $\langle y, m_1 \rangle - \langle y, m_2 \rangle = \langle y, m_1 - m_2 \rangle$; (3) $k \cdot \langle y, m \rangle = \langle y, k \cdot m \rangle$.

Operations of Sets of Resources. We define the type set $Y(R) = \{y \in Y : \exists m \in \mathbb{R}^+ : \langle y, m \rangle \in R\}$. We further define the operations (union, intersect, and minus) on set of resources: (1) $R_1 \tilde{\cup} R_2 = \varsigma(R_1 \cup R_2)$; (2) $R_1 \tilde{\cap} R_2 = \varsigma(R_1 \cap R_2)$; (3) $R_1 \tilde{\setminus} R_2 = \varsigma(R_1 \cup -R_2)$, where $-R = \{-r : r \in R\}$. The function ς is defined as $\varsigma(R) = \{r' \in \bigcup_{y \in Y(R)} \{\langle y, \sum_{r \in R, r.y = y} r.m \rangle\} : r'.m > 0\}$, which means that the resources with the same type ($r.y = y$) will be combined into one, and all the resources whose amount is below zero will be removed.

Required and Provided Resources. For each schedule σ, and task point $\pi \in \sigma$, we can define: (1) $R_{\triangleleft}(\pi)$ is the set of resources required by π, which is defined by $R_{\triangleleft}(\pi) = \pi.R_{\triangleleft}$; (2) $R_{\triangleright}(\pi)$ is the set of resources provided by σ after executing π, which is recursively defined by $R_{\triangleright}(\sigma(i)) = (R_{\triangleright}(\sigma(i-1)) \tilde{\cup} \pi.R_{+}) \tilde{\setminus} \pi.R_{-}$.

Resource Holder. This concept is introduced to uniformly analyse the change of the resource configuration of the entire system (including both the MRS and its environment). Each *resource holder* h is a tuple $\langle l, R, \sigma \rangle$, where (1) l is the location of h; (2) R is the set of resources currently possessed by h; (3) σ is the sequence of task points which are not yet executed and which will take effects on either the location l or the set of resources R after being executed. A holder h can be either a robot or a station. If h is a station, then its σ field is simply a representation of those resources scheduled to be given out to robots. If h is a robot, then its σ field shows the sequence of task points to be actually executed by the robot.

Rendezvous. Each *rendezvous* υ represents multiple robots meeting at a common time and location, and is defined as a tuple $\langle l, \Pi, H, p \rangle$, where (1) l is the location of the rendezvous; (2) Π is the set of task points attending υ; (3) H is the set of resource holders attending υ; (4) p is a function which corresponds each $h \in H$ to $\pi \in \Pi$, so that $p(h) = \pi$ shows that h attending υ through executing π.

Resource-delivering Rendezvous. In this work, all rendezvouses are used for delivering resources, and therefore should contain only two task points: (1)

π_{snd} for sending a set of resources R, which is executed by the resource provider; (2) π_{rcv} for receiving a set of resources R, which is executed by the robot demanding the resources. Thus for each resource-delivering rendezvous υ, there exists π_{snd}, π_{rcv}, and R, so that: $\upsilon.\Pi = \{\pi_{\mathsf{snd}}, \pi_{\mathsf{rcv}}\}$, $\upsilon.H = \{h_{\mathsf{snd}}, h_{\mathsf{rcv}}\}$, and $p = \{\langle h_{\mathsf{snd}}, \pi_{\mathsf{snd}}\rangle, \langle h_{\mathsf{rcv}}, \pi_{\mathsf{rcv}}\rangle\}$.

3.2 Rendezvous Tangling

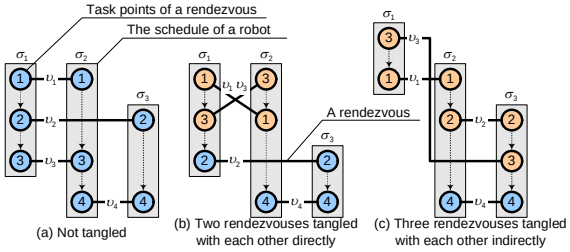

Figure 1: Tangling among robots.

Fig. 1 compares 3 cases related to deadlock among rendezvous: (1) **Not tangled** (Fig. 1-a). The task points of all rendezvouses can be completed without interfering with each other. No rendezvous-caused deadlock will occur. (2) **Directly tangled** (Fig. 1-b). In this case, we have

$$\exists \upsilon, \upsilon' \in \Upsilon : \exists \pi_1, \pi_2 \in \upsilon.\Pi, \pi'_1, \pi'_2 \in \upsilon'.\Pi$$
$$[\pi_1, \pi'_1 \in \sigma_1, \pi_2, \pi'_2 \in \sigma_2] :$$
$$\iota_{\sigma_1}(\pi_1) < \iota_{\sigma_1}(\pi'_1) \wedge \iota_{\sigma_2}(\pi_2) > \iota_{\sigma_2}(\pi'_2)$$

indicating a waiting loop that makes schedules not executable: (i) π_1 waits for π_2 (because $\pi_1, \pi_2 \in \upsilon.\Pi$); (ii) π_2 waits for π'_2 (because $\iota_{\sigma_2}(\pi_2) > \iota_{\sigma_2}(\pi'_2)$); (iii) π'_2 waits for π'_1 (because $\pi'_1, \pi'_2 \in \upsilon'.\Pi$); (iv) π'_1 waits for π_1 (because $\iota_{\sigma_1}(\pi_1) < \iota_{\sigma_1}(\pi'_1)$). (3) **Indirectly tangled** (Fig. 1-b). In this case, we have:

$$\exists \upsilon_1, ..., \upsilon_n \in \Upsilon :$$
$$\exists \pi_{1,1}, \pi_{1,2} \in \upsilon_1.\Pi, ..., \pi_{n,1}, \pi_{n,2} \in \upsilon_n.\Pi$$
$$[(\forall i \in \{2, ..., n\} : \pi_{i-1,2}, \pi_{i,1} \in \sigma_i) \wedge \pi_{n,2}, \pi_{1,1} \in \sigma_1] :$$
$$(\forall i \in \{2, ..., n\} : \iota_{\sigma_i}(\pi_{i-1,2}) < \iota_{\sigma_i}(\pi_{i,1})))$$
$$\wedge \iota_{\sigma_1}(\pi_{1,1}) > \iota_{\sigma_1}(\pi_{n,2})$$

indicating a waiting loop that makes schedules not executable: $\pi_{1,1}$ waits for $\pi_{n,2}$ which waits for $\pi_{n,1}$ which waits for $\pi_{n-1,2}, ..., \pi_{2,1}, \pi_{1,2}, \pi_{1,1}$.

3.3 Demand Looping

Assume initially the resource demands of all task points in the MRS are not satisfied. A straightforward way to satisfy these demands is to make each robot to traverse its entire schedule, where for each task point π whose resource demands $\pi.R_{\triangleleft}$ are not satisfied, the robot will generate a set of rendezvouses $\Upsilon_{\mathsf{prv}}(\pi)$ to fetch $\pi.R_{\triangleleft}$ from stations or other robots. If no inter-robot coordination strategy is present, an extreme situation known as *demand looping* may occur, in which robots will endlessly "argue" with each other about who should use the particular set of resources to satisfy itself, without reaching any conclusion. In this section we present two examples to show the need to introduce inter-robot coordination strategy so as to decide prudently (1) the insertion index of each rendezvous, and (2) the robot providing required resources; and the consequences of not to do so.

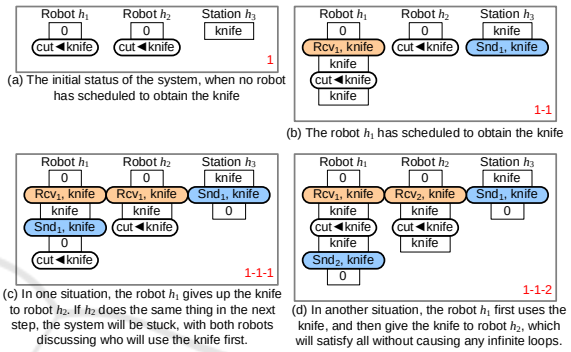

Figure 2: A simple case showing demand looping.

Fig. 2 shows the need to constrain the **insertion index of each rendezvous** among robots, in order to prevent the resource demands of one robot from being sabotaged by rendezvous with another robot. Fig. 2-a shows the initial state. Each robot has a task point for cutting with a knife. In Fig. 2-b, robot h_1 sets up the rendezvous for fetching the knife from station h_3. Since initially both robots possess no knife, and there is only one knife, intuitively, h_1 and h_2 must take turns using the knife, and therefore the ideal next step would be to let h_2 obtain the knife after h_1 has completed using it. However, due to the lack of coordination, in Fig. 2-c, h_2 abruptly schedules to take away the knife from h_1 without considering h_1's need, making h_1 unable to complete its schedule. If in the following steps, e.g., Fig. 2-d, they still repeat the same ruthless decision, then it is possible the rendezvouses for sending/receiving knife will be repeated for many times among h_1 and h_2, while the knife can never have the chance to be actually used. Note that in order to prevent this, we can forbid all robot from giving out resources that are already scheduled to be used by itself, i.e., to constrain the insertion of rendezvouses in the schedules of potential resource providers.

Fig. 3 shows that the MRS must be prudent in choosing the proper **resource provider and receiver** at each step. Fig. 3-a and Fig. 3-b descends from the same initial status when both robots have no resources and all resources are in h_3. Based on different choices

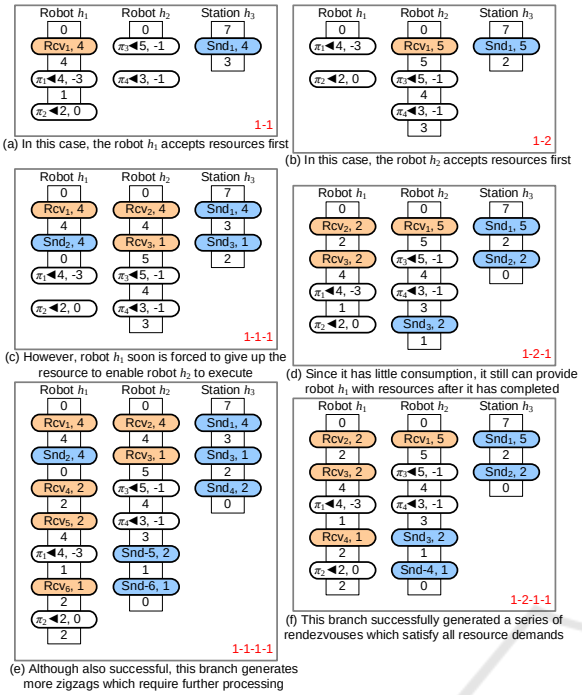

Figure 3: The case where initial choices are important.

made at the very first step about who should obtain the resources first, the system may undergo different journeys: (1) Fig. 3-a,c,e, shows the branch when h_1 first obtains the resources for its task point π_1; (2) Fig. 3-b,d,f, shows the branch when h_2 first obtains the resources for π_3. In Fig. 3-a,c,e, since satisfying π_1 has used up a lot of resources, leaving no resources for any other task points of h_1 and h_2, in Fig. 3-c, h_1 has no choice but to give up part of its resources and π_1, to let others run. In Fig. 3-b,d,f, even after all task points of h_2 are satisfied, there are still remaining resources in h_1, h_2, h_3, making it possible to satisfy h_1's demands. Although both branches can ultimately satisfying all the resources, Fig. 3-a,c,e branch creates some zigzags which involve robots planning to deliver resources forwards and backwards among them without using the resources, due to improper earlier choice of who would provide and/or receive the resources.

4 ANALYSIS AND SOLUTION

4.1 Prevention of Rendezvous Tangling

Reachable Set of a Rendezvous. We first define several functions based on reachable sets: (1) the directly preceding set of rendezvouses $\Upsilon_{\leftarrow}(\upsilon) = \bigcup_{\pi\in\upsilon.\Pi} \Pi_{\triangleleft}(\pi)$; (2) the directly succeeding set of rendezvouses $\Upsilon_{\rightarrow}(\upsilon) = \bigcup_{\pi\in\upsilon.\Pi} \Pi_{\triangleright}(\pi)$; (3) the preceding reachable set of rendezvouses is recursively defined by $\Upsilon_{\star\leftarrow}(\upsilon) \subseteq \Upsilon_{\star\leftarrow}(\upsilon)$ and $\forall\upsilon' \in \Upsilon_{\star\leftarrow}(\upsilon) : \Upsilon_{\leftarrow}(\upsilon') \subseteq \Upsilon_{\star\leftarrow}(\upsilon)$; (4) the succeeding reachable set of rendezvouses is recursively defined by $\Upsilon_{\rightarrow}(\upsilon) \subseteq \Upsilon_{\rightarrow\star}(\upsilon)$ and $\forall\upsilon' \in \Upsilon_{\rightarrow\star}(\upsilon) : \Upsilon_{\rightarrow}(\upsilon') \subseteq \Upsilon_{\rightarrow\star}(\upsilon)$. The ***no-tangling rule*** can be written as $\forall\upsilon \in \Upsilon : \Upsilon_{\star\leftarrow}(\upsilon) \cap \Upsilon_{\rightarrow\star}(\upsilon) = \emptyset$, which is expected to be satisfied by all the rendezvouses in the schedules at any given time.

In order to ensure there is no tangling, we constrain the insertion index of each rendezvous υ when it is inserted or moved in the schedules. For υ, if the insertion index $\iota_\sigma(\pi)$ of any $\pi \in \upsilon.\Pi$ has been specified, $\iota_\sigma(\pi)$ poses constraints to the possible insertion indexes of other task points in $\upsilon.\Pi$. Assume $\Pi_{\text{in}} \subseteq \upsilon.\Pi$ is the set of task points that have been inserted into the schedules without tangling. If we want to insert one more task point $\pi \in \upsilon.\Pi \setminus \Pi_{\text{in}}$ without introducing tangling, its insertion index i in schedule σ should satisfy the following constraint:

$$\max_{\upsilon'\in\Upsilon_{\star\leftarrow}(\upsilon)} \iota_\sigma(\pi') < i \leq \min_{\upsilon'\in\Upsilon_{\rightarrow\star}(\upsilon)} \iota_\sigma(\pi')$$

where $\{\pi'\} = \upsilon'.\Pi \cap \sigma \neq \emptyset$. It can be proven that as long as the insertion index of any task point of a rendezvous satisfy the above constraint, the no-tangling rule can always be ensured, and there will be no deadlock among robots caused by rendezvous tangling.

4.2 Prevention of Demand Looping

For preventing demand looping, all robots should collaborate to determine (1) which robot will generate the next rendezvous υ for satisfying one of its task points, (2) which resource holder will provide resource for υ. It should prevent the situation of having one robot generate all its rendezvouses at one shot without regarding other robot's demands. Also, each robot should consider the degree to which inserting υ may deprive the chance of the resource provider from further satisfying its own resource demands. For example, if by inserting υ, all further task points would lost their chances to obtain any resources from any resource holders, then definitely υ should not be inserted, provided there is a better choice.

Crosscut. In order to better depict such thought, we introduce the concept of *crosscut*. A crosscut is a function $\chi : H \rightarrow \mathbb{Z}$, which returns the currently concerned index in the schedule of each resource holder. Given a crosscut χ, (1) the set of resources provided by holder h is $R_{\triangleright}(h.\sigma(\chi(h)))$, which shows

from where other robots can obtain their required resources; (2) the set of resources required by holder h is $R_{\triangleleft}(h.\sigma(\chi(h)))$, which is used to compare to the provided resources $R_{\triangleright}(h.\sigma(\chi(h)))$ to see whether χ is unsatisfied; (3) the preceding crosscut of χ is written as $\chi-1$, which is defined as

$$(\chi-1)(h)=\begin{cases}\chi(h)-1 & \chi(h)>0\\ 0 & \text{otherwise}\end{cases}$$

(4) the succeeding crosscut of χ is written as $\chi+1$, which is defined as

$$(\chi+1)(h)=\begin{cases}\chi(h)+1 & \chi(h)<|h.\sigma|\\ \infty & \text{otherwise}\end{cases}$$

(5) we use $\chi=0$ to represent $\forall h\in H:\chi(h)=0$; (6) we use $\chi=\infty$ to represent $\forall h\in H:\chi(h)=\infty$.

Boundary Crosscut. We define *boundary crosscut* $\overline{\chi}$ for depicting the progress of rendezvous generation:

$$\forall h\in H:\quad \overline{\chi}(h)=\begin{cases} i & i\le|h.\sigma|\wedge h.\sigma(1,...,i-1)\text{ is satisfied}\\ & \quad\wedge\, h.\sigma(i)\text{ is not satisfied}\\ \infty & h.\sigma(1,...,|h.\sigma|)\text{ is satisfied}\end{cases}$$

The $\overline{\chi}$ is an upper limit beyond which all task points are not satisfied. With the rendezvous generation proceeding, we can imagine a great frontier of satisfied task points marked by $\overline{\chi}$ gradually pushing forwards and approaches the ends of all schedules (where $\overline{\chi}=\infty$), signifying the satisfaction of all task points.

We now make some rules about the process of approaching $\overline{\chi}=\infty$ in order to prevent demand looping.

1. The χ and $\overline{\chi}$ should advance to the goal $\overline{\chi}=\infty$, and should always choose the way to achieve that goal with lowest cost.
2. The insertion index of each rendezvous should be within $(\chi,\overline{\chi})$, so that the task points satisfied through previous rendezvous planning, should remain satisfied in the later rendezvous planning, and the planning will not omit any task point's demands nor sabotage any satisfied task points.
3. The task point to be satisfied in the next step of rendezvous planning should be chosen in such a way that maximises the opportunity of other task points to be satisfied in the future planning, so that entire MRS has the best chance to be all satisfied.

4.3 A*-based Rendezvous Planning

In this section we present a centralised A*-based Rendezvous Planning Algorithm (ARPA), which traverses all schedules to generate a set of rendezvouses that satisfy as many task points as possible. The deadlock prevention mechanisms shown in Section 4.1 and Section 4.2 are embedded in ARPA. ARPA can be further converted into an equivalent distributed version in which each robot processes its schedule while persistently communicating with other robots to obtain information for its own decision-making and to prevent deadlocks (of both types) among robots.

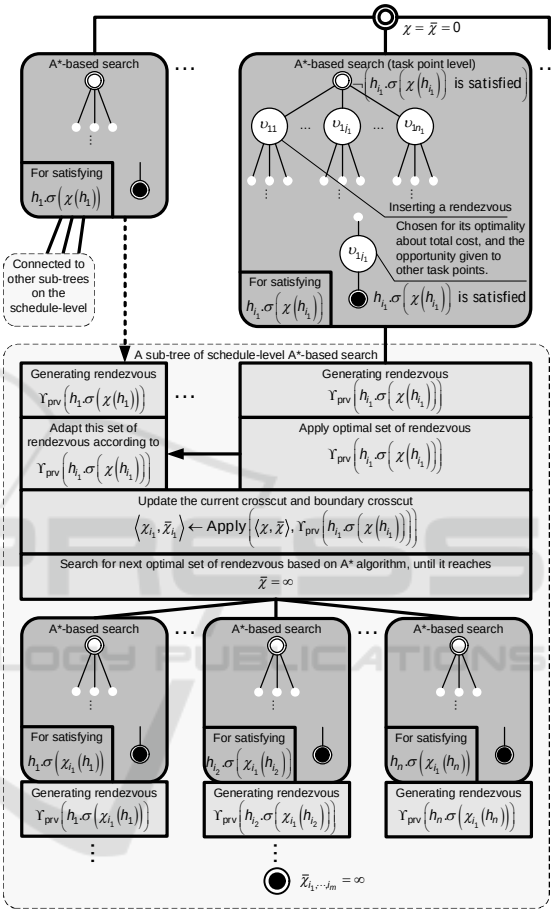

Figure 4: The search tree of A*-based rendezvous planning algorithm that combines two granularity levels.

Fig. 4 shows the search tree of ARPA. Assume initially all robots are without any resources, and all resources are in the fixed stations. The search begins with $\chi=\overline{\chi}=0$, to reach the goal $\overline{\chi}=\infty$. The search is divided into two granularity levels: (1) *schedule level*, which deals with the goal to make $\overline{\chi}=\infty$; (2) *task point level*, which deals with the sub-goal to satisfy the demands of each task point.

Schedule Level. In this level, A* algorithm is used to reach the goal $\overline{\chi}=\infty$. The algorithm will try to use the calculation at task point level to satisfy each task point π on $\overline{\chi}$, and generate the set of ren-

dezvouses $\Upsilon_{\mathsf{prv}}(\pi)$ which fetch resources for π. Based on $\{\Upsilon_{\mathsf{prv}}(\pi) : \pi \text{ is on } \overline{\chi}\}$, it finds the optimal task point π^*:

$$\pi^* = \arg\max_{\pi \text{ is on } \overline{\chi}} f_{\mathsf{s}}(\langle \Upsilon_{\mathsf{prv}}(\pi_1), ..., \Upsilon_{\mathsf{prv}}(\pi_n)\rangle, \Upsilon_{\mathsf{prv}}(\pi))$$

where $f_{\mathsf{s}} : (\mathfrak{P}(\Upsilon))^* \times \mathfrak{P}(\Upsilon) \to \mathbb{R}$ is a heuristic *fitness* function, which collectively considers following factors: (1) the cost of completing $\Upsilon_{\mathsf{prv}}(\pi)$; (2) the cost of previously applied $\langle \Upsilon_{\mathsf{prv}}(\pi_1), ..., \Upsilon_{\mathsf{prv}}(\pi_n)\rangle$; (3) the estimated distance from current search step to the goal $\overline{\chi} = \infty$, given the current χ and $\overline{\chi}$; (4) the estimated influence of applying $\Upsilon_{\mathsf{prv}}(\pi)$ on the opportunities of other task points becoming satisfied in the future. After π^* is chosen, it will be put into a frontier set χ which records all the optimal paths at the schedule level up to now, and will be the starting point of next round of schedule-level search. Each node ν_{s} of the search tree at the schedule level is defined as a tuple $\langle \pi, \chi, \chi', \Upsilon_{\mathsf{prv}}(\pi), \nu_{\leftarrow \mathsf{s}}\rangle$, where (1) π is the task point to be satisfied by $\Upsilon_{\mathsf{prv}}(\pi)$ (generated by the corresponding task point level search); (2) χ is the crosscut reached by ν_{s}; all task points before χ will not be modified by any searches succeeding ν_{s}; (3) χ' is the boundary crosscut reached by ν_{s}; (4) $\Upsilon_{\mathsf{prv}}(\pi)$ is the rendezvous sequence generated by the corresponding task point level search to satisfy $R_{\triangleleft}(\pi)$. (5) $\nu_{\leftarrow \mathsf{s}}$ is the node that precedes ν_{s} in the search tree, from which we can get the optimal path from $\chi = \overline{\chi}$ to ν_{s}.

Task Point Level. In this level, A* algorithm is used to satisfy $R_{\triangleleft}(\pi)$ — the demands of each task point π on $\overline{\chi}$ by finding the best rendezvous sequence $\Upsilon_{\mathsf{prv}}(\pi)$. The providing task point will be chosen from the section of crosscuts $(\chi, \overline{\chi})$. At each step, the algorithm will try to insert a new rendezvous into the $\Upsilon_{\mathsf{prv}}(\pi)$ until $R_{\triangleleft}(\pi)$ is fully satisfied. Similar to the schedule level, in this level, we also choose among all possible rendezvouses the optimal rendezvous υ^* based on the following formula:

$$\upsilon^* = \arg\max_{\upsilon \in \overline{\Upsilon}_{\mathsf{prv}}(\pi)} f_{\mathsf{t}}(\Upsilon_{\mathsf{prv}}(\pi), \upsilon)$$

where $f_{\mathsf{t}} : \Upsilon^* \times \Upsilon \to \mathbb{R}$ is the heuristic *fitness* function of rendezvous, which is defined similar to f_{s}. The most significant difference is that f_{t} considers the distance to the sub-goal where π is satisfied, rather than the satisfaction of entire schedule. Each node ν_{t} of the search tree at the task point level is defined as a tuple $\langle \upsilon, \chi, \overline{\chi}, \nu_{\leftarrow \mathsf{t}}\rangle$, where (1) υ is the rendezvous to be inserted after ν_{t} is chosen as optimal; (2) χ is the crosscut reached by ν_{t}; (3) $\overline{\chi}$ is the boundary crosscut reached by ν_{t}; (4) $\nu_{\leftarrow \mathsf{t}}$ is the node that precedes ν_{t} in the search tree, from which we can obtain the optimal path from $\Upsilon_{\mathsf{prv}}(\pi) = \emptyset$ to ν_{t}. The search will turn to the schedule level when the task point level search has completed for each task point in $\overline{\chi}$. In the task point level, when it is found that currently concerned task point cannot be satisfied without sabotaging other task points, the corresponding search will be postponed until next round of schedule level search.

5 CONCLUSION

In this paper we analyse the two types of deadlock that can be introduced into a multi-robot system (MRS) by uncoordinated generation of resource-delivering rendezvouses among robots. The first type is caused by rendezvous tangling and is tackled by constraining the insertion index of each rendezvous according to its preceding/succeeding reachable set. The second type is caused by one robot's resource demands being sabotaged by another robot's rendezvous, and is tackled by prudent selection of insertion index of each rendezvous and of resource provider/receiver when planning rendezvouses. We present the sketch of an A*-based Rendezvous Planning Algorithm (ARPA) that respects the constraints regarding both types of deadlock.

Our future work includes: (1) Determine the specific form of the fitness functions f_{s} and f_{t} as specified in Section 4.3. (2) Conduct an experiment which shows ARPA can generate efficient and effective rendezvouses for inter-robot delivery. (3) Implement a distributed version of ARPA with lower computation and communication cost, which facilitates its application on a real-world MRS. (4) Deal with the interaction between ARPA and the rescheduling and reallocating algorithms on MRS.

REFERENCES

Han Lei, Keyi Xing, L. H. F. X. Z. G. (2014). Deadlock-free scheduling for flexible manufacturing system using petri-nets and heursitic search. In *Computers and Industrial Engineering, v. 72, n. 1, p. 297-305*.

Keshmiri, S. (2011). Multi-robot, multi-rendezvous recharging paradigm: an opportunistic control strategy. In *IEEE ROSE*.

Neil Mathew, Stephen L. Smith, S. L. W. (2013). A graph-based approach to multi-robot rendezvous for recharging in persistent tasks. In *International Conference on Robotics and Automation*.

Neil Mathew, Stephen L. Smith, S. L. W. (2015). Multi-robot rendezvous planning for recharging in persistent tasks. In *IEEE Transaction on Robotics, v. 31, n. 1, p. 128-142*.