Manuscript
Click here to download Manuscript: paper.tex
Click here to view linked References

Noname manuscript No.
(will be inserted by the editor)

Communication Constrained Task Allocation with Optimized Local Task Swaps

Lantao Liu · Nathan Michael · Dylan A. Shell

Received: date / Accepted: date

Abstract Communication constraints dictated by hardware often require a multi-robot system to make decisions and take actions locally. Unfortunately, local knowledge may impose limits that ultimately impede global optimality in a decentralized optimization problem. This paper enhances a recent anytime optimal assignment method based on a task-swap mechanism, redesigning the algorithm to address task allocation problems in a decentralized fashion. We propose a fully decentralized approach that allows local search processes to execute concurrently while minimizing interactions amongst the processes, needing neither global broadcast nor a multi-hop communication protocol. The formulation is analyzed in a novel way using tools from group theory and optimization duality theory to show that the convergence of local searching processes is related to a shortest path routing problem on a graph subject to the network topology. Simulation results show that this fully decentralized method converges quickly while sacrificing little optimality.

Keywords Decentralized task allocation · Communication constraint · Task swaps · Permutation group

1 Introduction

Multi-robot task allocation or role assignment aims at finding the best matching between a set of robots and a set of tasks in order to optimize the team's performance. It is among the most popular optimization formulations for coordination problems in multi-robot systems. Solutions to general task allocation problems have also led to specialized methods for applications of particular importance in robotics research, *e.g.*, by strategically setting the tasks as goal locations, the methods efficiently deploy robots as part of path planning [Turpin et al., 2013, 2014] and formation control problems [Michael et al., 2008, Liu and Shell, 2012a]. Ultimately, a fundamental understanding of distributed assignment problems may also benefit other decentralized systems (or missions) such as automated transportation systems, large-scale swarm systems, unmanned planetary exploration, *etc.*

A multi-robot team may adapt to circumstances, demonstrating fluid coordination by repeatedly allocating tasks to robots frequently. While the classic optimal assignment (or weighted matching) algorithms seek to reduce overall execution time and time complexity, the multi-robot task allocation setting has several aspects that demand special consideration. For example, any central controller that is relied upon to compute and broadcast the assignment solutions becomes a single point of failure for the robot team. Instead, computation should be carried out in a distributed way so that individual failures do not affect the whole system. In addition to handling dynamics, emergencies, and unexpected contingencies smoothly, the responsiveness of the team depends on fast solution of the underlying assignment problem relative to the environment dynamics. But many envisioned scenarios involve multiple robots being dispatched in a large workspace where each robot may only be able to communicate with comparatively few neighbors who are nearby. Unfortunately long message relays may hinder the system's respon-

Lantao Liu, Nathan Michael
The Robotics Institute, Carnegie Mellon University
E-mail: {lantao, nmichael}@cmu.edu

Dylan A. Shell
Dept. of Computer Science and Engineering, Texas A&M University
E-mail: dshell@tamu.edu

siveness. Also, the optimality of an assignment solution becomes moot if the system's state evolves so rapidly that the decision was made with outdated inputs. Naturally both prohibitive communication delays and bandwidth limits may preclude the use of a *centralized* task allocation strategy.

The constraint of local communication may impact the final task allocation solution. If each robot is permitted to share information with only neighbors in its vicinity and no multi-hop communications stratagem is employed, the resultant assignment solution may necessarily be suboptimal. This is because a decentralized algorithm constrained by local communication cannot guarantee that complete information will be aggregated without relaying messages, whereas the search of optimal solution may require the complete information. Even for the simplest task allocation scenario involving a one-to-one mapping of robots to tasks, it is still unclear that under certain communication constraints, what the best (potentially sub-optimal) assignment can be obtained, and what the largest solution convergence step can be reached.

This paper explores decentralized task allocation methods where communication and computation are carried out concurrently in a spatially localized fashion so that the interaction between agents' local searching processes can be minimized.[†] The work extends the recent task-swap based anytime optimal assignment method [Liu and Shell, 2013], which has a decentralized structure but may still entail multi-hop communication (global information). In this paper, the same mechanism is responsible for the atomic optimization step (*viz.* the task-swap). But here we loosen the global communication requirement in the search procedure that selects which step is performed. Although the algorithm we present may produce suboptimal solutions—an inherent limitation arising from the fact that local information may be intrinsically inadequate—this new algorithm optimizes the search subject to the communication constraints and always produces the maximal step toward the optimal solution. Since local communication may impose limits that hinder global optimality, first we formulate a local optimality property and then we analyze it with duality theory and graph relaxation techniques. The decentralized nature of the method is shown using group theoretic notions.

[†]The main algorithm was first presented at the 2014 Robotics: Science and Systems conference [Liu et al., 2014].

2 Related Work

Many approaches to multi-robot coordination rely on task allocation to determine an efficient assignment of tasks to robots [Zlot and Stentz, 2006, Parker, 2008]. In a multi-robot system, robots need to not only take into account the presence of other team members but also to actively cooperate with them so as to achieve the best performance of the whole system. Different assignment models have arisen in order to formulate and address differing task allocation scenarios (see Gerkey and Matarić [2004] for a review). An important dimension within the task allocation taxonomy is the cardinality of the mapping between robots and tasks, *viz.*, whether the assignment relationship between robots and tasks is one-to-one, one-to-many, many-to-one, or many-to-many. In this work, we are interested in the problem of exclusively assigning every robot with a unique task (one-to-one mapping), which is the most fundamental and probably most widely investigated instance.

This class one-to-one mapping assignment problem is also termed the *linear sum assignment problem* [Burkard et al., 2009], and many optimal assignment algorithms have been developed in the last half century (see reviews [Pentico, 2007]). The majority are *primal-dual* methods (details of primal and dual formulations are presented in sections that follow). Important examples include the well-known Hungarian algorithm [Kuhn, 1955], which solves the matching problem by manipulating a matching bipartite graph. Several shortest augmenting path algorithms (*e.g.*, see Edmonds and Karp [1972], Derigs [1985]) were inspired by the Hungarian algorithm and also belong to primal-dual methods. Other notable assignment algorithms also include the Auction algorithm [Bertsekas, 1979], pseudo-flow [Goldberg and Kennedy, 1995] algorithms, *etc.* Most of these algorithms are capable of solving the problem with $O(n^3)$ time complexity when certain searching techniques and/or auxiliary data structures are employed.

However, one drawback of the classic optimal assignment algorithms is the difficulty of developing decentralized variants of these algorithms. This disadvantage becomes important for larger scale systems (involving tens, hundreds, even thousands of robots). Specifically, a centralized algorithm can result in large computation/communication imbalance since the central controller is usually responsible for collecting, processing, and disseminating information for the whole system. A heavy workload may result in low efficiency for the central controller, which yields low efficiency for the entire system. Also, a centralized coordination framework has poor robustness against failure owing to the dependence on the single central controller. These drawbacks also

apply to semi-centralized architectures with a limited fixed number of central controllers.

Several researchers have attempted to decentralize existing classical optimal assignment algorithms in order to apply them to distributed systems [Zavlanos et al., 2008, Giordani et al., 2010]. The resulting computational procedures have tended to reflect the strongly interconnected aspects implicit in the logical structure of the optimization problem, rather than the situational and spatial structure of the robot group. Many methods ignore the communication network topology, necessitating complex multi-hop communication protocols which involve the ability to route between arbitrary agents in the network. A major challenge lies in the fact that these algorithms tend to involve multiple phases or stages and there is a dependence between stages where one stage may rely on the outputs of those that precede it (similar but different from scheduling dependencies [Korsah et al., 2013]). While this coupling facilitates efficiency (the computation halts within strongly polynomial time/steps) it imposes strong synchrony requirements which are inimical to decentralization. An alternate framework, called *distributed constraint optimization (DCOP)* [Hirayama and Yokoo, 1997, Modi et al., 2006], considers a group of distributed agents which manipulate a set of variables such that the cost associated with a set of constraints over the variables is minimized. DCOP is NP-complete [Chechetka and Sycara, 2006], and many DCOP algorithms rely on preconstructed (global and static) tree structures, thereby failing to be robust against failures [Modi et al., 2006, Petcu and Faltings, 2005].

The multi-robot research community has also developed its own inherently decentralized approaches. An important set of these methods employ *market-based* [Dias et al., 2006, Tang and Parker, 2007] or *auction-based* mechanisms [Gerkey and Matarić, 2002, Lagoudakis et al., 2005] that emulate financial interactions between humans. Also important are *opportunistic* methods where pairs of robots within communication range adjust their workload by redistributing or exchanging tasks [Golfarelli et al., 1997, Lemaire et al., 2004]. Strategies using task switching [Sariel and Balch, 2006, Wawerla and Vaughan, 2009, Sung et al., 2013] or task exchanges [Chaimowicz et al., 2002, Farinelli et al., 2006] typically transfer tasks between pairs of robots whenever the operation improves the team's performance. These intuitively appealing methods allow for a form of localized, light-weight coordination of the flavor advocated by Stone et al. [2010]. The *task swap* mechanism generalizes the idea of pairwise task switches to larger cliques and has begun to be explored recently [Zheng and Koenig, 2009]. In these algorithms each robot has an assigned task at any moment so that the computation process can be interrupted at any time.

Prior to this work, Liu and Shell [2012b] proposed an assignment method based on task-swaps that is distributable, operates in an anytime fashion, and is optimal. It bridges the classical convex optimization view of optimal assignment problems, popular in operations research, with the decentralized task allocation approaches developed and favored by roboticists. Any stage of the algorithm's execution naturally involves only a subset of the robots and tasks, and communication with other parts of the system may not be required. But, despite its decentralized form, the method assumes that at each stage, the selected subset of robots *can* communicate with one another in some way, either via direct communication (when robots are within communication range, or when global broadcast communication exists) or a multi-hop network (if only local communication is available). This assumption may preclude use of the algorithm in scenarios where robots have only local communication and where not multi-hop communication protocol is available. We will show that this assumption is eliminated in this paper, and we propose a fully decentralized task allocation algorithm that minimizes interactions between robots via concurrent processes and always produces the maximal step toward the optimal solution.

The paper's organization is as follows: we first describe the problem and present necessary preliminaries in Section 3. In Section 4.2 we derive the permutation decomposition used for local strategic task swaps. The completely decentralized task swap method is detailed in Section 4. And finally, simulations of large-scale multi-robot task allocation Problems in Section 5 provide validation of the proposed method.

3 Problem Description and Preliminaries

We consider the multi-robot task assignment problem whose solution is an association of each robot to exactly one task, which has been termed the *single-task robots, single-robot tasks, instantaneous assignment* instance (ST-SR-IA) by Gerkey and Matarić [2004]. More formally, given a set of n available robots $R = \{r_1, r_2, \cdots, r_n\}$ and a set of n available tasks $T = \{t_1, t_2, \cdots, t_n\}$, and let $\mathbf{C} = (c_{ij})_{n\times n}$ be the *cost matrix*, where c_{ij} represents the cost of having robot i to perform task j, then the goal is to find a one-to-one mapping $\psi : T \to R$ that minimizes the overall cost. (Note, here we assume that the number of robots and number of tasks are equal; Scenarios with unequal number of robots and tasks are discussed in Section 4.3.)

3.1 Assignment Matrix, Permutation Cycle and Permutation Group

In this subsection, we describe terminologies and basic concepts that will be used for developing our decentralized method. Formulation of the assignment problem as matching or linear integer problems is well known [Burkard et al., 2009]. Unlike those popular treatments, we show later (Section 4) that the assignment may also be formulated using matrix and group operations. The primary utility of this approach is convenience in analyzing aspects related to decentralization of the algorithm.

3.1.1 Assignment Matrix

Let binary variable x_{ij} denote the assignment between robot task pair (i, j) so that x_{ij} is equal to 1 if assigned and 0 if unassigned, then an *assignment matrix* can be denoted as

$$X = \left(x_{ij}\right)_{n\times n} = \begin{pmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nn} \end{pmatrix}. \tag{1}$$

Since our assignment is a one-to-one mapping, in each row and each column of $\mathbf{X}$ there must be only one entry with value 1 and all others 0s.

3.1.2 Permutation Matrix and Permutation Cycle

Let $\mathbf{e}_k$ denote the vector of length n with 1 in the k^{th} position and 0 in every other positions, then we define the *permutation matrix* $\mathbf{P}$ as

$$\mathbf{P} = \begin{pmatrix} \mathbf{e}_{k_1} \\ \mathbf{e}_{k_2} \\ \vdots \\ \mathbf{e}_{k_n} \end{pmatrix}, \; k_i \neq k_j \text{ if } i \neq j. \tag{2}$$

Left (right) multiplying $\mathbf{X}$ by $\mathbf{P}$ reorders/permutes the rows (columns) of the assignment matrix. If only matrix entries are permuted while the row indices are fixed as $1, 2, \cdots, n$, the assignment solution is changed accordingly. For instance, using the prime to denote change (*not* matrix transpose):

$$\mathbf{PX} = \begin{pmatrix} 0&0&0&1 \\ 1&0&0&0 \\ 0&0&1&0 \\ 0&1&0&0 \end{pmatrix} \begin{pmatrix} 1&0&0&0 \\ 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \end{pmatrix} = \begin{pmatrix} 0&0&1&0 \\ 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \end{pmatrix} = \mathbf{X}'. \tag{3}$$

Equivalently, if we simplify the assignment matrix $\mathbf{X}$ to be a vector

$$\boldsymbol{\pi} = [\pi(1), \pi(2), \cdots, \pi(n)]^T \tag{4}$$

where $\pi(i)$ is the assigned task for robot i (*i.e.*, the index of elements in $\boldsymbol{\pi}$), then the example in Eq. (3) becomes

$$\mathbf{P}\boldsymbol{\pi} = \begin{pmatrix} 0&0&0&1 \\ 1&0&0&0 \\ 0&0&1&0 \\ 0&1&0&0 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 2 \\ 4 \end{pmatrix} = \boldsymbol{\pi}'. \tag{5}$$

This form helps reveal the changing of an assignment clearly. Comparing $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$, we can observe that the task t_1 (*i.e.*, number 1 in $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$) is transferred from robot r_1 to robot r_2 (*i.e.*, row index of t_1 in two vectors has changed from 1 to 2). Such a transfer operation is written as $\pi(1) \mapsto \pi(2)$;[‡]. Similarly, for robot r_2, we have $\pi(2) \mapsto \pi(4)$; and for robot r_4, we have $\pi(4) \mapsto \pi(1)$. Robot r_3, having no change, keeps task t_2. For the three robots that have changed tasks, a cycle has formed between them:

$$\pi(1) \mapsto \pi(2) \mapsto \pi(4) \mapsto \pi(1). \tag{6}$$

Definition 1 A *Permutation Cycle* with length K is an ordered chain of K distinct elements

$$i_1 \mapsto i_2 \mapsto \cdots \mapsto i_K \mapsto i_1, \tag{7}$$

where i_k denotes the index of the elements. In our context, i_k can be regarded as the index of robots, and the cycle may be imagined as robots passing tasks to their successors. See Fig. 1 for an illustration.

We write the cycle in Eq. (6) as (124)(3) or simply (124) since the single-element cycle (3) is an identity map. Additionally, a cycle of length 2 is termed a *transposition* (two robots exchanging tasks).

Definition 2 *Disjoint Cycles* are different cycles that do not share a common element.

Note that a cycle is *directed* and the element positions are not commutative, but the cycle notation is not unique since any form connecting the head and tail represents the same cycle, *e.g.*, (124) = (241) = (412) but (124) ≠ (142).

[‡] Note that this notation represents a change from our conference paper [Liu et al., 2014]. The arrow direction in the present work better illustrates the underlying task-exchange operator.

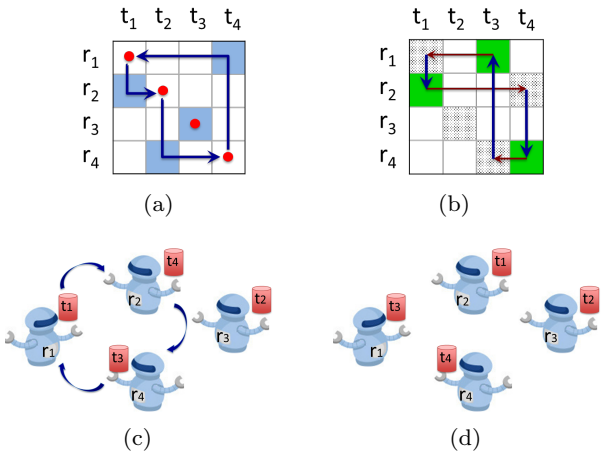


Fig. 1 Illustration for Eq. (3)–(6). (a) A permutation cycle in the permutation matrix $\mathbf{P}$. The rows with dark entries that are off the diagonal shall be permuted. Diagonal dark entries represent identity maps; (b) In the assignment matrix, a new assignment solution substitutes the old one. The dot-textured entries denote old assignment $\mathbf{X}$, whereas the solid dark entries are the new candidates obtained by $\mathbf{X}^{'} = \mathbf{PX}$. A horizontal arrow represents the substitution of new task for a robot, whereas a vertical arrow denotes that for a specific task, its owner is changed from one robot to another; (c) Robots r_1, r_2 and r_4 exchanged their tasks along a closed cycle (124), *e.g.*, r_1 passes its task to r_2 after permutation; (d) New task assignment result.

3.1.3 Permutation Group

Definition 3 A *group* $(\mathcal{G}, *)$ consists of a nonempty set $\mathcal{G}$ together with a binary operation $*$ on $\mathcal{G}$ satisfying the following conditions:

1) (*Closure*): $a * b \in \mathcal{G},\ \forall a, b \in \mathcal{G}$;
2) (*Associativity*): $(a * b) * c = a * (b * c),\ \forall a, b, c \in \mathcal{G}$;
3) (*Identity element*): $\exists e \in \mathcal{G}, a * e = a = e * a,\ \forall a \in \mathcal{G}$;
4) (*Inverse element*): $\exists a^{-1} \in \mathcal{G}, a * a^{-1} = e = a^{-1} * a,\ \forall a \in \mathcal{G}$.

In our assignment problem, the composition of two bijections always gives another bijection, the product of two permutations is again a permutation. Consequently, the set S_n of all permutations of $R = T = \{1, 2, ..., n\}$ (for simplicity, we denote the robot and task IDs with numerical symbols) forms a *permutation group* with operations given by composition, viewing permutations as functions from $R(= T)$ to itself.

Let $S_n = (\mathcal{G}, *)$ denote the permutation group with

$$\mathcal{G} = \{g_1, g_2, \cdots, g_m\} \tag{8}$$

where g_i is a cyclic permutation and operator $*$ is multiplicative. Then a series of permutations on assignment $\boldsymbol{\pi}$ can be written as

$$(g_i(g_j(\cdots(g_k\boldsymbol{\pi}))) = (g_i g_j \cdots g_k)\boldsymbol{\pi} = g'\boldsymbol{\pi}, \quad g' \in \mathcal{G}, \tag{9}$$

where the multiplicative binary operator $*$ is omitted.

There are two important propositions that are related to this work:

Proposition 1 *Every permutation in S_n can be written as a product of disjoint cycles. Disjoint cyclic permutations are subject to the commutative law [Rotman, 1995].*

Remark: The commutative property indicates that disjoint permutation cycles can be executed in an arbitrary order, which, expresses the essential orderless and executional independence within a decentralized implementation.

Proposition 2 *Any permutation cycle can be written as a product of transpositions (cycles of length 2) [Rotman, 1995].*

$$(i_1 i_2 \cdots i_k) = (i_1 i_2)(i_2 i_3) \cdots (i_{k-1} i_k) \tag{10}$$

However, the transpositions are not disjoint and thus not commutative.

Remark: This can be understood by considering the permutation as a *bubble sorting* algorithm which swaps positions of two elements each time.

By further extending these basic operations, we show that the permutation group can be effectively used to minimize interactions among local task allocation processes.

3.2 Assignment Optimization

Now, considering the optimization perspective, the assignment problem can be formulated so as to relate to a pair of linear programs.

One is a cost minimization formulation called the *primal program* $\mathcal{P}(R, T)$:

$$\begin{aligned} \text{minimize } & f(R,T) = \sum_{i \in R, j \in T} c_{ij} x_{ij}, \\ \text{subject to } & \sum_{j \in T} x_{ij} = 1, \quad \forall i \in R, \\ & \sum_{i \in R} x_{ij} = 1, \quad \forall j \in T, \\ & x_{ij} \geq 0, \quad \forall i \in R, j \in T, \end{aligned} \tag{11}$$

where each x_{ij} represents a primal variable. As a consequence of the structure of the constraint matrix, the problem turns into an integer problem and eventually

each x_{ij} will equal 0 or 1 in the solution when solved via some combinatorial optimization algorithm. The constraints $\sum_j x_{ij} = 1$ and $\sum_i x_{ij} = 1$ guarantee that no two robots are assigned with the same task and no two tasks are allocated to the same robot.

There are corresponding *dual* vectors $\mathbf{u} = \{u_i\}$ and $\mathbf{v} = \{v_j\}$ in the *dual program* $\mathcal{D}(R,T)$:

$$\begin{aligned} &\text{maximize } h(R,T) = \sum_{i\in R} u_i + \sum_{j\in T} v_j, \\ &\text{subject to } \ u_i + v_j \le c_{ij}, \quad \forall i \in R, j \in T. \end{aligned} \tag{12}$$

The dual program solves the problem from a complementary perspective: the sum of each pair (u_i, v_j) can be interpreted as the collective profit that a robot-task pair (r_i, t_j) gains, and the objective of the program then is to maximize the overall profit.

Theorem 1 *(The Duality Theorem [George, 1963]) If two programs $\mathcal{P}(R,T)$ and $\mathcal{D}(R,T)$ are feasible, then $f(R,T) \ge h(R,T)$. If either program has a finite optimal value, then so does the other, and the optimal values satisfy $f^*(R,T) = h^*(R,T)$.*

Remark: Given finite cost values, our assignment problem always produces a finite optimal value. Then the theorem points to three requirements for the existence of the optimal solution: (i) feasibility of $\mathcal{P}(R,T)$; (ii) feasibility of $\mathcal{D}(R,T)$; (iii) $f(R,T) = h(R,T)$.

Maintaining (i) and (ii) is straightforward, but directly reaching the condition of (iii) is not. In fact, by assuming $f(R,T) = h(R,T)$, and adjusting $\mathcal{P}(R,T)$ and $\mathcal{D}(R,T)$, the following result is obtained:

Theorem 2 *(The Complementary Slackness Theorem [George, 1963]) The optimal solution exists if and only if x_{ij} are feasible for $\mathcal{P}(R,T)$ and u, v are feasible for $\mathcal{D}(R,T)$, and*

$$x_{ij}(c_{ij} - u_i - v_j) = 0, \quad \forall i \in R, j \in T. \tag{13}$$

Remark: Eq. (13) reveals the property of orthogonality between the primal variables and reduced costs. It also indicates that if a robot-task pair (i,j) is assigned, *i.e.*, $x_{ij} = 1$, then the corresponding reduced cost $\bar{c}_{ij}$ must be equal to 0, where

Definition 4 *Reduced costs* are defined as

$$\bar{c}_{ij} = c_{ij} - u_i - v_j, \quad \forall i \in R, j \in T. \tag{14}$$

The constraint in Program (12) implies that only if $\bar{c}_{ij} \ge 0$ will the robot-task pair (i,j) be *feasible*. In essence, the reduced cost is an auxiliary variable used to describe the feasibility of pairwise dual variables between robots and tasks (*i.e.*, we use one variable to describe feasibility of two dual variables).

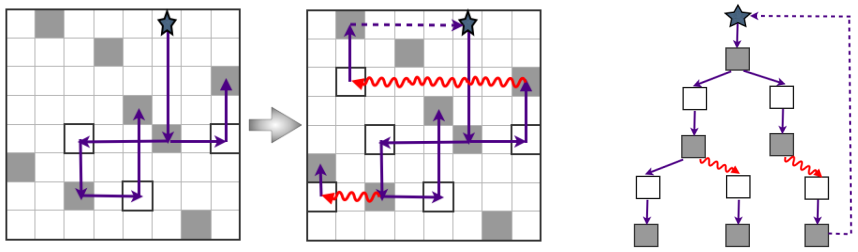
3.3 Strategic Task Swap Method

The task swaps in our previous work [Liu and Shell, 2012b] can be regarded as cyclic permutations. The permutation cycle is termed the *swap loop* or *swap cycle* in the task allocation context. A swap loop differs from a permutation cycle in that the swap loop is associated with two types of objects (robots and tasks), and are generated in a rather more strategic way. Fig. 1(a) shows a permutation cycle (124) where the arrows hop over multiple rows $1 \to 2 \to 4$, crossing three dark cells. In contrast, the cycle in Fig. 1(b) is a swap loop involving two types of dark entries (and six dark cells are crossed). A swap loop can be transformed to a permutation cycle by keeping only one type of entry in each row: only the robots/rows information need be retained in the cycle notation. To update the assignment, each robot on a swap loop substitutes its task with its successor's along the closed orbit.

Our prior task-swap assignment algorithm [Liu and Shell, 2012b] essentially built upon the *primal*-based algorithm of Balinski and Gomory [1964]. The algorithm maintains a feasible primal (requirement (i) in the Duality Theorem) and the complementary slackness (equivalent to requirement (iii)), and iteratively adjusts the dual program (requirement (ii)). The optimal solution is determined when the dual also becomes feasible. In the centralized context, it was shown that with a time complexity of $O(n^3 \lg n)$ and a maximum number of $O(n)$ swap loops, the optimal assignment solution is guaranteed to be found.

The main steps appear in Algorithm 3.1 as pseudocode. At a high-level, a spanning tree-like data structure (in the reduced cost matrix) is used to search for swap loops. See Fig. 2 for an illustration. Searching starts from an entry (i_0, j_0) with infeasible reduced cost ($\bar{c}_{i_0 j_0} < 0$). In Fig. 2(a), the entry containing a star is such an entry. New entries are added as tree nodes by establishing traversal edges (links). Specifically, if the current leaf node is an assigned entry with $x_{ij} = 1$, it is expanded to new leaf nodes that are unvisited unassigned entries satisfying $\bar{c}_{ij} = 0$ in the same row; otherwise if the current leaf node is unassigned, it is expanded to the unique assigned entry in the column that it resides in. Two sets R_v, T_v are used to record the expanded/visited rows and columns, respectively. Let i' be the row of an assigned leaf node, then

$$\begin{aligned} &\textit{Update } R_v, T_v \textit{ for } i' : \\ &\qquad R_v = R_v \cup \{i'\}, \\ &\qquad T_v = T_v \cup \{j' \mid \bar{c}_{i'j'} = 0, \forall j' \in T \setminus T_v\}. \end{aligned} \tag{15}$$



(a) (b)

Fig. 2 Illustration of swap loop searching in a reduced cost matrix. (a) From a starting entry (the one that contains a star), new entries are added as tree nodes by establishing traversal edges (links). Shaded entries are currently assigned, and bold-edged entries have reduced costs equal to zero. Waved lines represent the paths found after dual adjustments; (b) A corresponding spanning-tree like data structure is used to aid efficient searching.

This searching procedure repeats until a loop is found — when a leaf node hits the starting row i_0. (Note that, $i_0 \notin R_v$.)

However, if no entry with 0-valued reduced cost can be found before a loop is formed, the dual variables are then adjusted to introduce new entries with a reduced cost of 0.

Adjust $\mathbf{u}, \mathbf{v}$:

$$\begin{aligned} \delta &= \min \{\bar{c}_{ij} \mid i \in R_v, j \in T \setminus T_v\} \\ u_i &= u_i + \delta, \quad \forall i \in R_v, \\ v_j &= v_j - \delta, \quad \forall j \in T_v. \end{aligned} \tag{16}$$

A swap loop L can also, therefore, be thought of as a chain of entries alternatively satisfying $x_{ij} = 1$ and $x_{i'j'} = 0$, and all entries on the loop satisfying $\bar{c}_{ij} = 0$ (the starting infeasible entry (i_0, j_0) is an exception). A task swap operation is a substitution of x_{ij} by $x_{i'j'}$. We compare the old and new solutions:

$$\begin{aligned} & f(\mathbf{X}') - f(\mathbf{X}) \\ &= \sum_{i \in R, j \in T} x'_{ij} c'_{ij} - \sum_{i \in R, j \in T} x_{ij} c_{ij} = \sum_{(i,j) \in L} c'_{ij} - \sum_{(i,j) \in L} c_{ij} \\ &= \sum_{(i,j) \in L} (c'_{ij} - (u_i + \delta) - (v_j - \delta)) - \sum_{(i,j) \in L} (c_{ij} - u_i - v_j) \\ &= \sum_{(i,j) \in L} \bar{c}'_{ij} - \sum_{(i,j) \in L} \bar{c}_{ij} = \sum_{(i,j) \in L} \bar{c}'_{ij} = \bar{c}'_{i_0 j_0} \end{aligned} \tag{17}$$

The new solution is improved if $\bar{c}'_{i_0 j_0} < 0$, then the overall cost is reduced by an amount of $|\bar{c}'_{i_0 j_0}|$ after swapping tasks along the loop.

Additional detail on Algorithm 3.1 can be found in Liu and Shell [2012b].

Algorithm 3.1 Centralized Task Swap Algorithm

1: */* i, j are indices of rows and columns, respectively. $\pi^{-1}(\cdot)$ denotes the inverse of an assignment. */*
2: Initialize: $\mathbf{u} = \mathbf{0}, \mathbf{v} = \text{diag}(\mathbf{C})$
3: **for** $j = 1 \to n$ **do**
4: Get the smallest entry: $(i_0, j_0) = \text{argmin}_i\{\bar{c}_{ij}\}$; if $\bar{c}_{i_0 j_0} > 0$, break *for* loop
5: Queue $Q \leftarrow (i_0, j_0)$
6: **while** Q is not empty **do**
7: Q pops the top node, assuming entry (i_t, j_t); locate a new row $i' = \pi^{-1}(j_t)$
8: $Q \leftarrow \{(i', j') | \bar{c}_{i'j'} = 0, \forall j' \in T \setminus T_v\}$
9: Update sets R_v, T_v for i' via Eq. (15)
10: **if** $i' == i_0$ **then** a swap loop is formed, terminate
11: **if** Q is empty and loop is not formed **then**
12: Adjust $\mathbf{u}, \mathbf{v}$ via Eq. (16), new entries with $\bar{c}_{i'j'} = 0$, $i' \in R \setminus R_v$, $j' \in T \setminus T_v$ must exist
13: Go to Step 8

4 Decentralization of Task Swaps under Communication Constraint

The preceding formulation in Section 3.3 has potential for a decentralization because each stage (*i.e.*, the periodic process of finding one swap loop) is likely to involve only a subset of the robots. But some challenges must still be overcome.

- First, there is the question of how the robots gain access to the required information. The constraints imposed by the communication network affect the search process because, in order to reach an optimal solution, the algorithm requires that all robots in the spanning tree be able to reach one another. Assuming that this will always be possible is unrealistic for multi-robot systems with limited communication capabilities, unfortunately.
- The second issue lies in inter-stage dependencies, *i.e.*, a robot cannot be involved in two spanning trees simultaneously because the iterative dual updates must proceed sequentially.

We propose the following method to address the problems above.

4.1 Building Spanning Trees while Cognizant of the Communication Topology

We first consider the case of building a single spanning tree to search for only one swap loop, but subject to local communication. The goal is to span a search tree directly on the network topology rather than on the reduced cost matrix. As already mentioned, the swap loop that achieves the most substantial improvement in costs will be sought, but it can only be found subject to the constraints on information that may be transmitted.

We start by analyzing the reduced costs on the swap loop and have the following lemma.

Lemma 1 *Let $\bar{c}_{ij}$ be the reduced costs of the* unassigned *entries (i,j) with $x_{ij}=0$ on the swap loop L, and define the sum c_L as*

$$c_L = \sum_{(i,j)\in L, x_{ij}=0} \bar{c}_{ij}, \tag{18}$$

then c_L is the difference between the new and old solutions:

$$c_L = f(\mathbf{X}') - f(\mathbf{X}). \tag{19}$$

Proof To show this we proceed from the right side to the left. From Eq. (17), we have

$$f(\mathbf{X}') - f(\mathbf{X}) = \bar{c}'_{i_0 j_0} = c_{i_0 j_0} - u'_{i_0} - v'_{j_0}. \tag{20}$$

Since $j_0 \in T_v$ but $i_0 \notin R_v$, Eq. (16) implies that u'_{i_0} is never updated. Assuming the searching carries out a sequence of dual updates $\delta = \{\delta_1, \delta_2, \cdots, \delta_l\}$, then,

$$\begin{aligned} &f(\mathbf{X}') - f(\mathbf{X}) = c_{i_0 j_0} - u_{i_0} - v_{j'_0} \\ =&c_{i_0 j_0} - u_{i_0} - (v_{j_0} - (\delta_1 + \delta_2 + \cdots + \delta_l)) \\ =&\bar{c}_{i_0 j_0} + (\delta_1 + \delta_2 + \cdots + \delta_l), \end{aligned} \tag{21}$$

where $\{\delta_1, \delta_2, \cdots, \delta_l\}$ are exactly those unassigned $\bar{c}_{ij}$ (except the starting entry) on the swap loop. Let $c_P = \delta_1 + \delta_2 + \cdots + \delta_l$, then we have

$$\begin{aligned} &f(\mathbf{X}') - f(\mathbf{X}) = \bar{c}_{i_0 j_0} + c_P \\ =&\bar{c}_{i_0 j_0} + \sum_{(i,j)\in L\setminus\{(i_0,j_0)\}, x_{ij}=0} \bar{c}_{ij} \\ =&\sum_{(i,j)\in L, x_{ij}=0} \bar{c}_{ij} = c_L. \end{aligned} \tag{22}$$

Lemma 1 reveals that only unassigned reduced costs can effect improvements to the solution and, except $\bar{c}_{i_0 j_0}$, all other reduced costs on the loop are positive. With this observation, we construct a graph $G=(V,E)$ where each robot and its assigned task are collected into a super node $v_\alpha = (r_\alpha \leftrightarrow \pi(r_\alpha)) \in V$, and we reinterpret those feasible (positive valued) unassigned entries as edges* $e=(v_\alpha, v_\beta) \in E$ with the corresponding reduced costs as edge weights $w(v_\alpha, v_\beta) = \bar{c}_{r_\alpha \pi(r_\beta)}$. Note, $w(v_\alpha, v_\beta) \neq w(v_\beta, v_\alpha)$, thus $e(v_\alpha, v_\beta) \neq e(v_\beta, v_\alpha)$. This model leads immediately to the following theorem.

Theorem 3 *The problem of searching for a task swap loop is transformed into a search for a cycle on a standard directed graph $G=(V,E)$, and where the total cycle weight c_L is exactly the cost reduction between new and old assignment solutions.*

In actuality graph G is built from the multi-robot communication network topology. Fig. 3 illustrates an example. The solid edges in the figures represent the connectivity of the network with edge weights as the corresponding feasible reduced costs. The dashed edges connecting to the starting node $\bar{c}_{r_1 t_1}$ (shown as leftmost) represent the infeasible entries which are not really on the graph, but they will be used to close the loop. The spanning tree grows by appending new edges (those with the least weight) and c_P is the path cost between the root and a leaf node on the tree. Finally, an dashed edge with weight -7 closes the loop consisting of r_1, r_2, r_3. The overall cost is reduced by $-c_L = -(-7 + c_P) = 2$ after the tasks are swapped.

4.1.1 Refining Swap Loop Searching via Relaxation

Further observation permits an improvement to the local search approach just described.

Theorem 4 *The spanning tree approach of Algorithm 3.1 is a* greedy *search method and the swap loop it finds may have an cost step size that is suboptimal, i.e., the reduction in optimization objective is not guaranteed to be maximal.*

Proof In each iteration of Algorithm 3.1, only nodes that are connected to the tree (but not yet on the tree itself) will be considered. And those nodes with minimum connecting edge weight, there may be more than one, will be added as new leaf nodes. The tree always employs the locally optimal choice first and grows in a way analogous to a breath-first-search (BFS). The searching process is, therefore, greedy. A path from root to a leaf node on a spanning tree (even a minimum spanning tree) need not necessarily be the shortest path, which would be the path that reduces the cost maximally.

* The symbols u and v will be used as vertices of graphs in the remainder of the paper. They are not to be confused with their use earlier as dual variables.

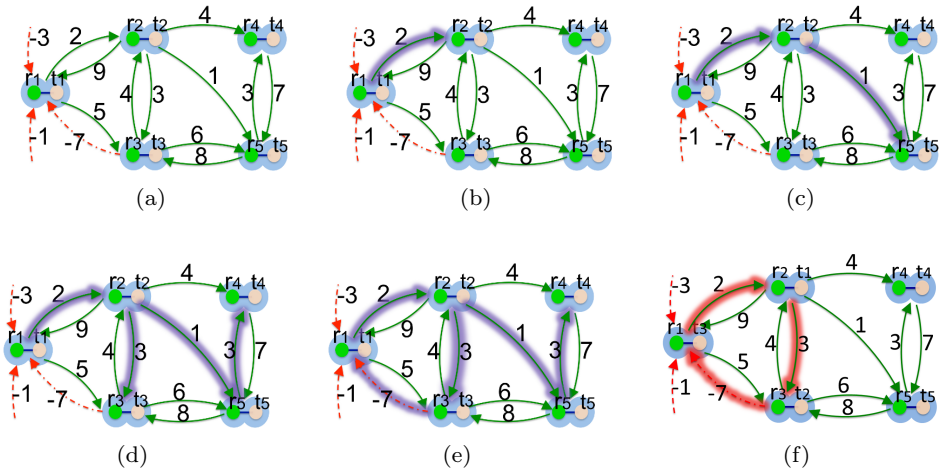


Fig. 3 A swap loop is found on the graph built on network topology. Solid edges with $w(v_\alpha, v_\beta) = \bar{c}_{r_\alpha \pi(r_\beta)} > 0$ represent the connectivity of the network. The dashed edges connecting to the leftmost node (the starting node) are not actually part of the graph, but represent infeasible entries. (a)—(e) The spanning tree, shown with thick edges, grows. (e) An infeasible edge closes the loop with $c_P = 2 + 3 = 5$ and $c_L = -7 + c_P = -2$. (f) After swapping tasks, the assigned robot-task pairs in the super nodes are changed. The connectivity of the graph is also updated.

The proof of Theorem 4 sheds light on directions for improvement of the local search. Lemma 1 shows that improvement of the assignment solution is essentially determined by the path cost c_P. We can improve the path quality using the *relaxation* technique popularly employed in single-source-shortest-path algorithms such as Dijkstra's algorithm. The steps (in pseudo-code) appear as part of in Algorithm 4.1.

Algorithm 4.1 Swap Loop with Relaxation

1: */* Let $v.d$ be the distance from node v to root; $v.d$ is ∞ by default.*/*
2: Initialize the root node v_0 s.t. $v_0.d = 0$
3: Set $S = \varnothing$, min priority $Q_p \leftarrow v_0$
4: **while** Loop is not found **do**
5: $\quad Q_p$ pops the top node, assuming v; $S = S \cup \{v\}$
6: $\quad$ **for each** node u on an outgoing edge of v **do**
7: $\quad\quad$ **if** $u.d > v.d + w(u, v)$ **then**
8: $\quad\quad\quad u.d = v.d + w(u, v)$, set u's predecessor as v
9: $\quad\quad\quad$ If between the root and a leaf node there exists a path P with $c_P + \bar{c}_{i_0 j_0} < 0$, a swap loop is formed

Relaxation aims at decreasing the cost of reaching a node by using another node adjacent to it. Unlike a spanning tree method, the edges among leaf nodes can be used to find loops.

When a node is relaxed (Step 6–8), the path to it is shorter and its predecessor is also modified so that a smaller number of nodes/robots may be involved in the path. Also, a swap loop with $c_L < 0$ may appear earlier, reducing the communication load through an early termination.

Additionally, since the relaxation compares nodes in set S and those outside S, once a node is in S, the path from it to the root must be the shortest. However, Step 9 implies that a loop can be detected even before the relaxation is finished. The algorithm can stop at any point in time after a swap loop is detected, any additional iterations refine the relaxation and can improve quality. Once all the nodes in the loop are in set S, the swap loop must converge to that which reduces the cost the most.

The time complexity for searching for a swap loop is improved over Algorithm 3.1. Specifically, given a total of n nodes/robots (n can be the size of either the whole system in the centralized version, or partial system in the decentralized version), building a spanning tree with Algorithm 4.1 requires only $O(|E|+|V|\lg|V|)$, which is $O(n^2)$ in the worse case (the analysis is analogous to the Dijkstra's algorithm); in contrast, Algorithm 3.1 needs $O(n^2 \lg n)$ to finish the spanning tree starting from the same root [Liu and Shell, 2012b].

4.2 Eliminating Inter-Stage Dependencies via Permutation Cycle Decomposition

In this section, we propose a means of mitigating the dependencies between the algorithm stages associated with multiple spanning trees. We will show that the dependencies among task swaps, originally presented in our previous work, can be greatly relaxed through a mechanism of permutation cycle decomposition, thereby requiring only local communication and fully distributing the method.

Based on Propositions 1 and 2, we have the following observations:

Lemma 2 *Any permutation cycle with length greater than or equal to three can be decomposed into* non-disjoint *permutations of shorter lengths.*

Proof Given an arbitrary permutation g with length $k \geq 3$, one can decompose it into two smaller cycles at element i_p in the original cycle,

$$\begin{aligned} g &= (i_1 i_2 \cdots i_k) \\ &= (i_1 i_2) \cdots (i_{p-1} i_p)(i_p i_{p+1}) \cdots (i_{k-1} i_k) \\ &= \big((i_1 i_2) \cdots (i_{p-1} i_p)\big)\big((i_p i_{p+1}) \cdots (i_{k-1} i_k)\big) \\ &= (i_1 i_2 \cdots i_p)(i_p \cdots i_k). \end{aligned} \tag{23}$$

Such a decomposition can occur on any element in the cycle so long as $1 < p < k$.

Note that the smaller cycles obtained in Eq. (23) do not commute because the element i_p is involved in two resulting non-disjoint cycles, which must be executed strictly in order if one is to get a result identical to the original cycle. To mitigate this strong ordering dependence, we have the following theorem.

Theorem 5 *If two non-disjoint permutation cycles g_1 and g_2 share a common element i_p, then i_p can be isolated from g_1 and g_2 through further decompositions. The resultant two new cycles g_1' and g_2' (without i_p involved) become disjoint depending on an appropriate adjustment of the remainder cycles with length at most 3.*

Proof Following the notation in Lemma 2, let $g_1 \overset{\text{def}}{=} (i_1 i_2 \cdots i_p)$ and $g_2 \overset{\text{def}}{=} (i_p \cdots i_k)$, the common element is thus i_p. It is worth noting that, since i_p is involved in two cycles, and in each cycle it has two neighbors (one predecessor and one successor), thus i_p has totally four neighbors (i_1, i_{p-1} in g_1, and i_{p+1}, i_k in g_2) that it directly connects to.

Now we decompose the two permutations:

$$\begin{aligned} & g_1 g_2 \\ =& (i_1 \cdots i_{p-1} i_p)(i_p i_{p+1} \cdots i_k) \\ =& (i_1 \cdots i_{p-1})(i_{p-1} i_p)(i_p i_{p+1})(i_{p+1} \cdots i_k) \\ =& (i_1 \cdots i_{p-1})(i_{p-1} i_p i_{p+1})(i_{p+1} \cdots i_k) \\ \overset{\text{def}}{=}& g_1' \tilde{g} g_2'. \end{aligned} \tag{24}$$

It shows that two cycles become three cycles where the middle one $\tilde{g} = (i_{p-1} i_p i_{p+1})$ has length 3. Neither of the other two ending cycles g_1' and g_2' contain i_p and are thus disjoint.

The ending cycles g_1' and g_2' can commute and switch order only if appropriate operations are carried out. More formally, let $\hat{g}$, $\check{g}$ be two transpositions formed by i_p and its four immediate neighbors, *i.e.*, $\hat{g} \overset{\text{def}}{=} (i_k i_1)$ and $\check{g} \overset{\text{def}}{=} (i_{p-1} i_p)$. If we attach $\hat{g}, \check{g}$ each after the commuted ending cycles, then we have:

$$\begin{aligned} & (g_2' \hat{g})(g_1' \check{g}) \\ =& (i_{p+1} \cdots i_k)(i_k i_1)(i_1 \cdots i_{p-1})(i_{p-1} i_p) \\ =& (i_{p+1} \cdots i_k i_1)(i_1 \cdots i_{p-1} i_p) \\ =& (i_{p+1} \cdots i_k i_1 \cdots i_p) \\ =& (i_1 i_2 \cdots i_k) = g. \end{aligned} \tag{25}$$

From Eq. (23) and (24), $g = g_1 g_2 = g_1' \tilde{g} g_2'$, thus Eq. (25) indicates that the two decomposed ending cycles g_1' and g_2' can commute after two transpositions $\hat{g}, \check{g}$ are performed after each.

Note that, each of the remainder cycles $\tilde{g}, \hat{g}, \check{g}$ involves at most 3 elements and represents local operations near i_p. One may regard the above manipulation as the operation of stitching smaller non-disjoint cycles—which reflect operations that can be computed locally and concurrently—into a larger one, involving a larger number of simultaneously acting robots. This operation leads to a greater degree of centralization for a potentially better solution.

4.2.1 Minimization of Local Search Interactions

We minimize interactions among agents' local searching processes (and augment assignment solution quality) through manipulating decomposed permutation cycles. More specifically, since both Algorithm 3.1 and 4.1 assume that during the searching procedure the task assignment information remains unchanged, this assumption can be violated if a robot is involved in multiple spanning trees because executing a loop formed earlier inevitably assigns this robot a new task, possibly causing later loops to be invalid (*i.e.*, $c_L > 0$ and no longer improving the assignment solution).

This means that a fully decentralized approach with the fewest interactions is desired. In other words, we prefer the local search processes to execute concurrently while minimizing interactions amongst the processes. We combine the permutation manipulations to explore the decentralized structure, and carefully examine interactions between the search procedures which assess the tasks to be swapped.

In greater detail, the analysis of the decomposition of permutation cycles in Theorem 5 shows that non-disjoint cycles can be decomposed into two disjoint ones with small remainder cycles. We use this observation to coordinate multiple non-disjoint task swap loops.

Assume task swapping along a loop

$$L : (\cdots i_{p-1} i_p i_{p+1} \cdots) \tag{26}$$

is executed first, and a common robot i_p is also involved in the loop formed thereafter

$$L' : (\cdots i'_{p-1} i_p i'_{p+1} \cdots). \tag{27}$$

In this case, $\tilde{g} = (i_{p-1} i_p i'_{p+1})$, $\hat{g} = (i'_{p-1} i_{p+1})$ and $\check{g} = (i_{p-1} i_p)$.

We first check the validity of the later loop L' by comparing the changes in reduced costs that are associated with i_p. Only valid loop will proceed to be refined. Let j_p denote the original task for i_p before L is executed, and $j_p = \pi(i_p)$ denote its updated task due to L, then loop L' is valid if and only if

$$\bar{c}_{i_p j_p} - \bar{c}_{i_p j'_p} < c'_L, \tag{28}$$

where c'_L is the total cost of loop L' as defined before. (More generally, if two loops share multiple common robots, the changes in reduced costs associated with these common robots are summed up and compared with c'_L.)

If the later loop L' is valid, we then further augment the solution by "stitching" L and L'. Since there are two ways of combining the two loops, *i.e.*, we can either put L first which corresponds to Eq. (24) (in this case $\tilde{g}$ needs to be performed) or put L' first as Eq. (25) describes (in this case $\hat{g}$ should be carried out). This can be addressed by comparing between operations of $\tilde{g}$ and $\hat{g}$, and the valid/best operation is selected to perform immediately after the first loop. Similarly, $\check{g}$ is also tested after executing the second loop and is performed if it can further improve the solution quality. (As we mentioned earlier, robots involved in $\tilde{g}, \hat{g}$ and $\check{g}$ can be easily located since they are adjacent in the loop and are neighbors in the network.)

4.3 Decentralized Algorithm with Concurrent Loop Searching

Thus far, the two aforementioned hurdles for decentralization—global communication requirement and stage dependency—have been eliminated. As a consequence, multiple swap loops can be searched concurrently and any robot is allowed to participate in multiple searching processes simultaneously. Robots communicate with neighbors by passing messages containing the latest spanning tree information. Note that the robot that initiated the loop (the root of the search tree) does not need to explicitly communicate with the last robot on the tail (a leaf of the search tree) in order to close the loop. This is because a loop closure can always be detected by the last robot on the chain and the task swaps can be done via backtracking along the loop.

In the decentralized implementation each robot carries out Algorithm 4.1 to maximize the local solution quality. The whole algorithm appears in Algorithm 4.2.

Algorithm 4.2 Decentralized Implementation

```
/* Each robot i maintains a record of neighbors N(i) and their assignment information. */
Select the root: e.g., root i' can be voted among neighbors by (i', j') = argmin_(i,j) c̄_ij, ∀i ∈ N(i), j ∈ T
Each root starts spanning tree on communication graph G
for each robot i do
    if robot i is root of some tree then
        if A (optimal) loop is formed then
            Revisit those in-loop robots to check loop validity
            if loop is valid then
                Notify other robots on the same tree to stop spanning
                Execute task swaps following the loop
                Stitch loops by selecting g̃, ĝ, ǧ
    else
        if i receives a message then
            Span and relax the tree following Algorithm 4.1
            if the path P_i to i satisfies c̄_{i'j'} + c_{P_i} > 0 then
                Stop spanning from this node
```

Finally, the algorithm does not need to use global information since the graph is only built from the network topology, however, the solution may be suboptimal because the network imposes constraints which confine the search so it will be an incomplete search of the global solution space. Any resulting suboptimality reflects the constraints imposed by the network connectivity.

Theorem 6 *Algorithm 3.1 yields a result that is optimal when executed on robots with a communication network that is a complete graph.*

Proof Since the optimization problem solved (in either primal or dual form) is convex, when any pair of robots can communicate with one another, the problem does not involve any local minima. Lemma 1 implies that when the relaxation runs to completeness, the largest cost reduction is found (otherwise Dijkstra's algorithm would be suboptimal). This means that if some progress toward a solution can be made then it must be found by the search and, without local minima, optimality must result.

We have not discussed the task allocation scenarios with unequal number of robots and tasks until now. There are two cases: (1) The number of robots is greater than the number of tasks. In this case, dummy/virtual

tasks with sufficiently large costs can be created for the initial assignment, so that the problem is transformed to the classic one-to-one mapping assignment problem. (2) The number of robots is less than that of the tasks. In this case, directly creating dummy robots does not work since dummy robots will not be able to do any computation or be involved in communication. However, the presented algorithm will still converge even with fewer number of robots. This is because in our approach each robot builds search trees and interacts with neighbors locally, dynamically and concurrently, in other words, the algorithm is not sensitive to the number of robots since to each robot only nearby neighboring robots matter, so long as each robot can hold a task at any time.

5 Experiments

We tested the proposed algorithm in simulation to validate our claims of improved local searching, fast convergence, and low communication load. Data were generated through a dispatching scenario: a group of robots in the plane must visit a set of destination way-points. Costs are computed as the Euclidean distances between the pair-wise robots and destination points. This setting was selected for its straightforward comprehension and in order to introduce as few domain specific complexities as possible.

Fig. 4 illustrates an example configuration generated with 50 robots and 50 target points randomly distributed in a $100m \times 100m$ square. Light line segments denote the spanning tree edges, and swap loops are drawn with thicker lines. Note that the searching process requires the underlying graph to be static. This strong assumption is exactly the motivation for designing a decentralized method with local searching, fast convergence, and anytime output. Fig. 4(a) also shows that in order to find a swap loop, only a subset of robots (and their tasks) need to be involved. Fig. 4(b) shows the corresponding swap loops in the reduced cost matrix.

5.1 Optimized Local Searching

Local searching with relaxation improves the greedy BFS search described in [Liu and Shell, 2012b]. Fig. 5(a) shows the practical running time in order to search for a swap loop. We can see that the time required for the relaxation method is much less than that of the BFS approach especially when the system is large. (Experiments were run on a standard laptop of dual-core CPU (2GHz×2) and 2GB memory, and all statistics are the mean values of 100 sets of data.)

Fig. 5(b) is a representative example showing that the assignment solution evolves faster with relaxation. The stairs in Fig. 5(b) show the decreasing objective value $f(\mathbf{X})$, where each decrement results from task swaps along a swap loop. This difference manifests itself as a larger downward step in cost reduction for the relaxation method.

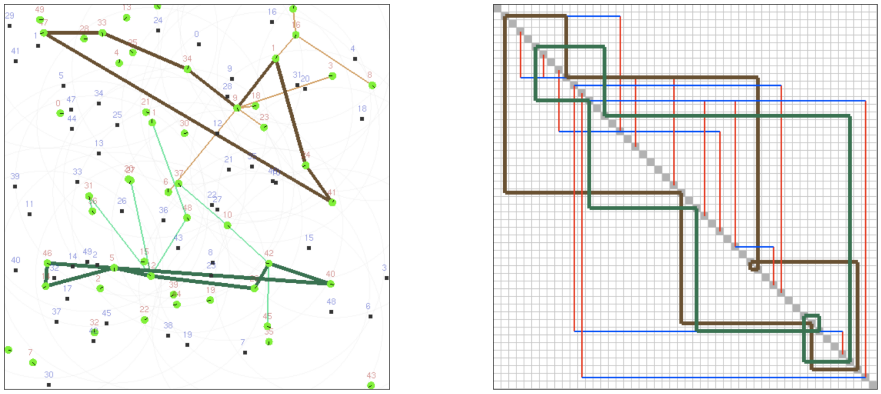
5.2 Solution Convergence

We next compared the convergence performance of the decentralized implementation (Algorithm 4.2) with the recent task-swap based optimal assignment algorithm [Liu and Shell, 2012b] that also allows interruptions at any time. We define each time step as an interval that allows a message to be sent or received, and assume that a robot is able to duplicate messages when necessary and broadcast them to multiple receivers in a time step (similar to the communication strategy of MURDOCH architecture [Gerkey and Matarić, 2002]).

The trade-off between the optimality and decentralization is revealed in Fig. 6(a), where we observe that the optimal algorithm with stage dependence and global communication has a linear trend reaching the optimum, whereas the decentralized approaches with limited communication produces inferior results. It should be emphasized, however, that without a multi-hop protocol the decentralized approach has limited information that can be shared; it is solving a more constrained optimization problem. Note also that owing to the local search being concurrent, the convergence rate of the decentralized is much faster than that of the centralized method. Specifically, under the same communication radius ($= 20m$), both the relaxation and greedy BFS converge rapidly, but the relaxation method has a solution quality significantly improved over the greedy BFS strategy. Then for the relaxation method, we enlarged the communication radius to $30m$, and observed that its convergence rate decreases whereas the solution (Relaxation+) is even closer to the optimum, which demonstrates Theorem 6.

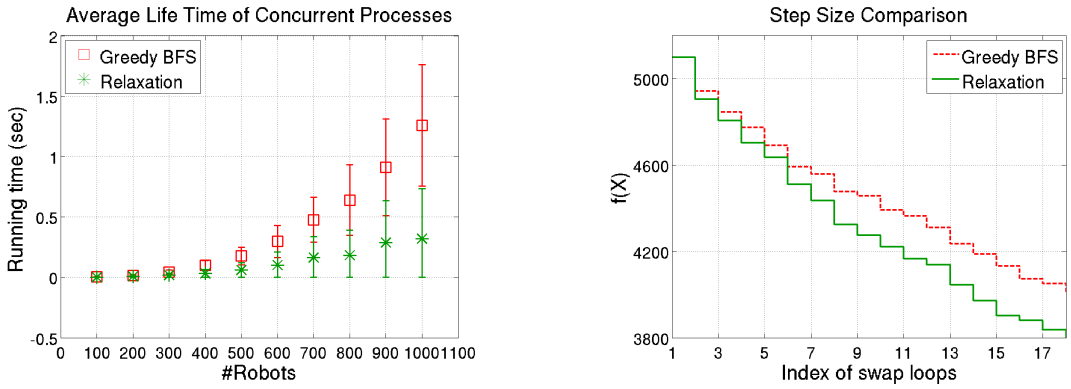
The number of processes used while searching was also varied in order to investigate its impact. Fig. 6(b) shows that with increased concurrency, convergence is quicker and the solution quality is improved. It also shows that the margin of improvement diminishes with increasing size, presumably reflecting the degree of parallelism intrinsic to the problem.

We opted to use a distributed constraint optimization (DCOP) algorithm for further comparison: since



(a) (b)

Fig. 4 Concurrent searching for swap loops. (a) The light-green circles represent robots and the smaller square dots denote tasks, where tasks can be mobile targets. Note that the longest edge on each loop is a virtual edge that closes the loop. In practice, a loop closure can always be detected by the last robot on the chain and the task swaps can be done via backtracking along the loop. (b) Permutation cycles in the reduced cost matrix. A dark entry (i, j) represents the allocation of robot i to task j. The initial assignment is arbitrary: a robot is assigned to the tasks with the same ID.



(a) (b)

Fig. 5 Local searching with optimized steps. With the relaxation described in Algorithm 4.1, (a) the practical running time is improved; (b) the step size is optimized .

our method dynamically constructs search trees, the asynchronous distributed constraint optimization (ADOPT) [Modi et al., 2006] method, which also utilizes a search tree structure, was selected for evaluation. (ADOPT has been used as a benchmark to compare many DCOP problems [Chechetka and Sycara, 2006, Yeoh et al., 2008].) ADOPT requires that a global search tree is constructed in which the agents each form tree nodes. Each agent i holds and controls a unique assignment variable x_i where $x_i = 1, \cdots, n$ maps to the assigned task (randomly chosen at first). To avoid global communication, we assume that x_i can exchange the values/tasks with only those agents in the same sub-tree. Then ADOPT controls the messages analogous to the branching technique in the Branch and Bound methods, but with special treatment and in a distributed fashion [Modi et al., 2006]. A demonstration of our implementation is shown in Fig. 7.

Fig. 6(c) shows that DCOP can converge to global optimal solution when the tree is maintained throughout the whole process (*i.e.*, it remains static). However, if the global tree is broken into multiple smaller trees (e.g., owing to agent failures), then the performance deteriorates drastically. Dependence on a static global tree allows DCOP to reach only a certain level of decentralization, and it falls short of what we consider fully distributed. Our method does not suffer from above issue because trees are created and destroyed dynamically and locally.

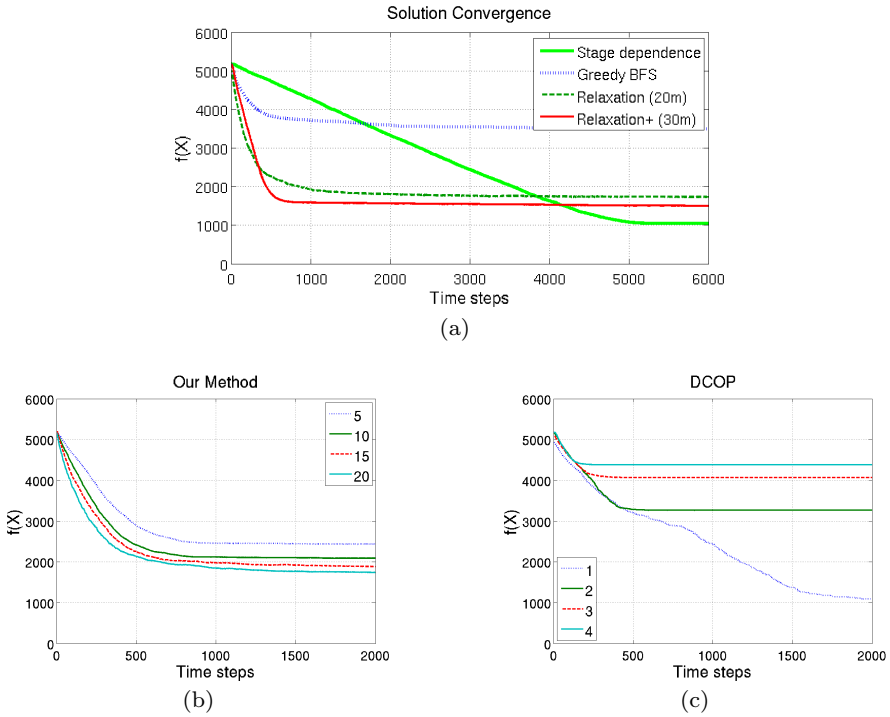


Fig. 6 Solution convergence analysis. (a) The trade-off between the convergence and the optimality for centralized, greedy BFS and relaxation methods of different communication radii (100 robots); (b) Our method: performance under different numbers of searching processes (5–20); (c) DCOP: performance under different numbers of static search trees (1–4).

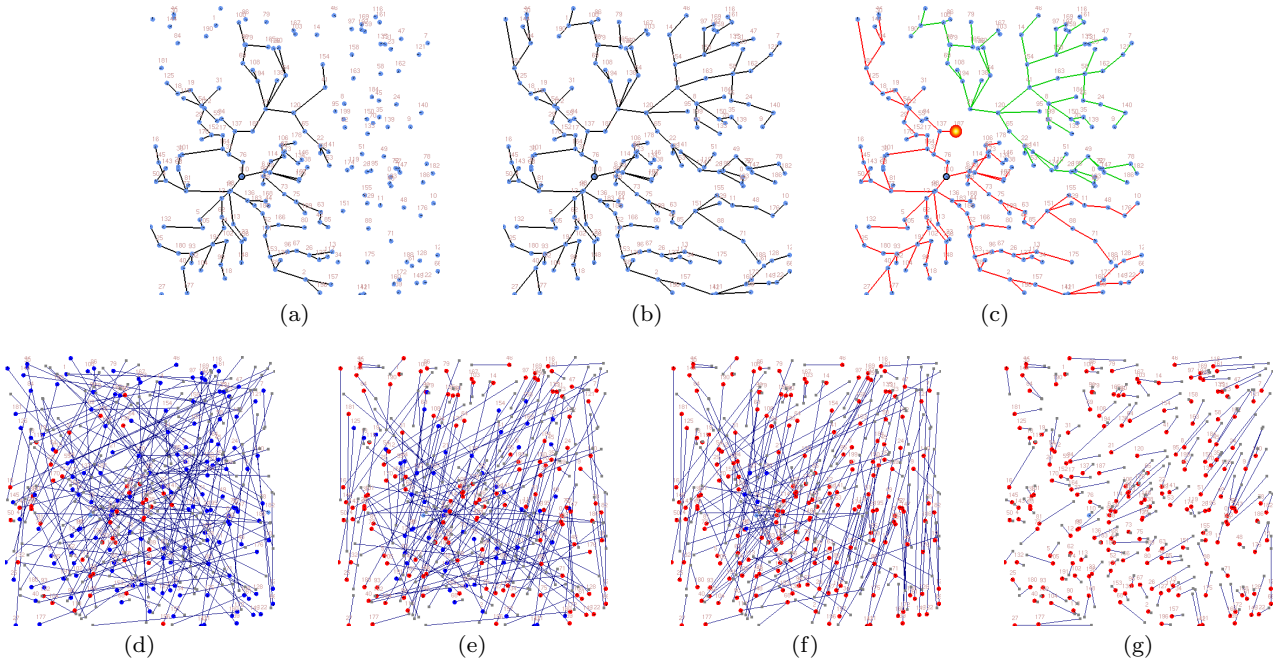


Fig. 7 Task allocation formulated with DCOP. (a) Global tree construction; (b) A complete tree connecting all agents; (c) Two disjoint sub-trees resulted from the failure of an agent; (d-g) Evolution of assignment matching along with DCOP operations (a red agent represents its sub-tree is optimal whereas a blue agent represents its sub-tree is still sub-optimal); (g) The optimal solution from a complete tree.

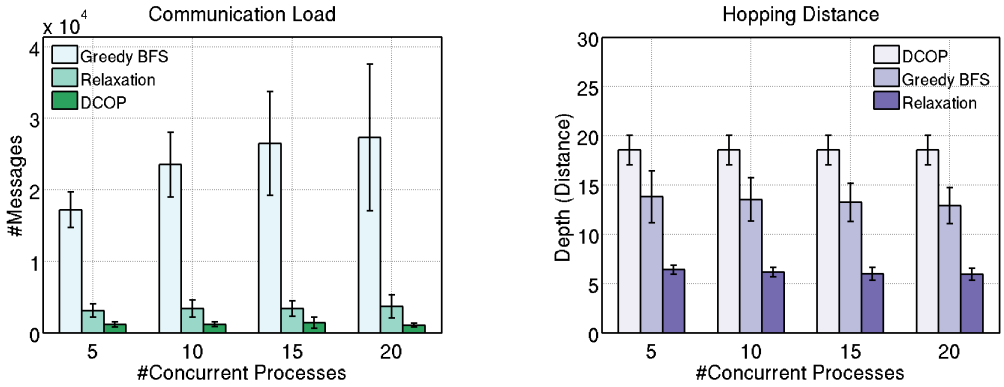


Fig. 8 Communication comparison in terms of (a) the total number of messages and (b) the average longest hopping distance under different number of concurrent processes.

5.3 Communication Analysis

Performance in terms of communication costs for the decentralized implementations was also assessed and compared. A measure of communication load is constructed by counting the total number of messages transmitted across the whole system. Since different methods converge at different rates and eventually reach different qualities, we thus define the communication load as the total number of messages that are used to decrease $f(\mathbf{X})$ by a fixed amount. Fig. 8(a) plots the communication needed to reduce $f(\mathbf{X})$ by 1000 from the initial (random) solution. We can see that the communication load is reduced significantly when the relaxation is employed instead of the greedy BFS. It also shows that the DCOP approach requires the least communication since all its messages flow on a global search tree which remains static.

Finally we investigated properties of the spanning trees, where the tree depth reflects the longest message passing (hopping) distance needed to find a swap loop. Longer hopping distances may cause longer time delays and are likely to involve more distant robots, both of which reflect a deterioration of the decentralization. Fig. 8(b) shows that the relaxation reduces the tree depth by more than half when compared with greedy BFS; in contrast, the DCOP also needs more total hops since the global tree has long branches.

6 Conclusion

We propose a new fully decentralized task swap based multi-robot task allocation method that respects single-hop communication constraints. The approach allows concurrent searching processes to proceed locally on a graph built over the network topology, and the interactions among local processes are minimized. Our formulation of the problem draws on techniques from group theoretic concepts and optimization duality theory to gain insight into the process of searching within a local subspace of a global optimization problem. We are able to connect the optimization convergence step size to the quality of a shortest path problem on a graph. Our simulation results show that this fully decentralized method converges quickly without sacrificing much optimality, despite we feel that the constraints imposed by a limited locus of knowledge and a global optimality criterion are essentially oppositional aspects.

References

M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10(3):578–593, 1964.

D. P. Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Report, MIT*, 1979.

R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems.* Society for Industrial and Applied Mathematics, New York, NY, 2009.

L. Chaimowicz, M. F. M. Campos, and V. Kumar. Dynamic role assignment for cooperative robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 293–298, 2002.

A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1427 – 1429, May 2006.

U. Derigs. The Shortest Augmenting Path Method for Solving Assignment Problems–Motivation and Com-

putational Experience. *Annals of OPerations Research*, pages 57–102, 1985.

M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM 19(2):248–264*, 19(2):248–264, 1972.

A. Farinelli, L. Iocchi, D. Nardi, and V. A. Ziparo. Assignment of dynamically perceived tasks by token passing in multi-robot systems. In *Proc. of the IEEE, Special Issue on Multi-robot Systems*, 2006.

D. George. *Linear Programming and Extensions*. Princeton University Press, August 1963.

B. P. Gerkey and M. J. Matarić. Sold!: auction methods for multirobot coordination. *IEEE Trans. on Robotics and Autom.*, 18(5), 2002.

B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23 (9):939–954, September 2004.

S. Giordani, M. Lujak, and F. Martinelli. A Distributed Algorithm for the Multi-Robot Task Allocation Problem. *LNCS: Trends in Applied Intelligent Systems*, 6096:721–730, 2010.

A. V. Goldberg and R. Kennedy. An Efficient Cost Scaling Algorithm for the Assignment Problem. *Math. Program.*, 71(2):153–177, 1995.

M. Golfarelli, D. Maio, and S. Rizzi. Multi-agent path planning based on task-swap negotiation. In *Proc. UK Planning and Scheduling Special Interest Group Workshop*, pages 69–82, 1997.

K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

G. A. Korsah, A. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *International Journal of Robotics Research*, 32(12):1495–1512, 2013.

H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistic Quarterly 2:83–97*, 2:83–97, 1955.

M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems*, 2005.

T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-UAV context. In *Proc. ICRA*, pages 3622–3627, 2004.

L. Liu and D. A. Shell. Multi-robot formation morphing through matching graph. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2012a.

L. Liu and D. A. Shell. A distributable and computation-flexible assignment algorithm: From local task swapping to global optimality. In *Proceedings of Robotics: Science and Systems*, 2012b.

L. Liu and D. A. Shell. An anytime assignment algorithm: From local task swapping to global optimality. *Auton. Robots*, 2013.

L. Liu, N. Michael, and D. Shell. Fully decentralized task swaps with optimized local searching. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas. Distributed multi-robot task assignment and formation control. In *IEEE Intl. Conf on Robotics and Automation*, pages 128–133, 2008.

P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2006.

L. E. Parker. Multiple Mobile Robot Systems. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, chapter 40. Springer, 2008.

D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, pages 774–793, 2007.

A. Petcu and B. Faltings. A scalable method for multi-agent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 266–271, 2005.

J. J. Rotman. *An introduction to the theory of groups*, volume 148. Springer Science & Business Media, 1995.

S. Sariel and T. Balch. A distributed multi-robot cooperation framework for real time task achievement. In *Proceedings of Distributed Autonomous Robotic Systems*, 2006.

P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *Proc. AAAI*, 2010.

C. Sung, N. Ayanian, and D. Rus. Improving the performance of multi-robot systems by task switching. In *IEEE International Conference on Robotics and Automation*, pages 2984 – 2991, 2013.

F. Tang and L. E. Parker. A Complete Methodology for Generating Multi-robot Task Solutions Using ASyMTRe-D and Market-based Task Allocation. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA'93)*, pages 3351–3358, 2007.

M. Turpin, K. Mohta, N. Michael, and V. Kumar. Goal assignment and trajectory planning for large teams

of aerial robots. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

M. Turpin, N. Michael, and V. Kumar. CAPT: Concurrent assignment and planning of trajectories for multiple robots. *International Journal of Robotics Research*, 33(1):98–112, 2014.

J. Wawerla and R. T. Vaughan. Robot task switching under diminishing returns. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 5033–5038, 2009.

W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. In *In Proceedings of AAMAS*, pages 591–598, 2008.

M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas. A Distributed Auction Algorithm for the Assignment Problem. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1212–1217, Cancun, Mexico, December 2008.

X. Zheng and S. Koenig. K-swaps: cooperative negotiation for solving task-allocation problems. In *Proc. IJCAI*, pages 373–378, 2009.

R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *I. J. Robotic Res.*, 25(1): 73–101, 2006.