Fully Decentralized Task Swaps with Optimized Local Searching

Lantao Liu
Robotics Institute
Carnegie Mellon University
lantao@cmu.edu

Nathan Michael
Robotics Institute
Carnegie Mellon University
nmichael@cmu.edu

Dylan A. Shell
Dept. of Computer Science and Engineering
Texas A&M University
dshell@cse.tamu.edu

***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 run against global optimality in a decentralized optimization problem. This paper redesigns the task-swap mechanism recently introduced in an anytime assignment algorithm to tackle the problem of decentralized task allocation for large scale multi-robot systems. 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 the 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.**

I. Introduction

Multi-robot task allocation or 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 one of the most popular optimization formulations for coordination problems in multi-robot systems. Solutions to general task allocation problems have also lead 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 [25, 26] and formation control problems [17, 15]. Ultimately, a fundamental understanding of distributed assignment problems may also benefit other decentralized systems/missions such as automated transportation systems, large-scale swarm systems, unmanned planetary exploration, *etc*.

A multi-robot team may adapt to circumstances and demonstrate fluid coordination by allocating tasks to robots repeatedly. 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 nearby neighbors. Consequently, long message relays may hinder the system responsiveness. 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.

Several researchers have attempted to decentralize existing classical optimal assignment algorithms in order to apply them to distributed systems [28, 10]. Because the computation exploits the logical structure of these optimization problems rather than the situational and spatial structure of the group, the computational procedures remain strongly coupled. 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. One 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 precedes it (similar but different from the scheduling dependence [13]). While this coupling facilitates efficiency (the computation halts within strongly polynomial time/steps) it imposes strong informational synchrony which is inimical to decentralization. An alternate framework, called *distributed constraint optimization (DCOP)* [12, 18], 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 [4], and many DCOP algorithms rely on pre-constructed (global and static) tree structures, thereby failing to be robust against failures [18, 19].

The multi-robot research community has also developed its own inherently decentralized approaches. An important set of these methods employ *market-based* [6, 23] or *auction-based* mechanisms [8, 14], that emulate financial interactions between humans. These methods employ a form of localized, light-weight coordination similar in some regards to that advocated by [21]. Although some theoretical upper and lower performance bounds are known for very basic auction based strategies [14], analysis of solution quality is generally rare for such methods.

Also important are *opportunistic* methods where pairs of robots within communication range adjust their workload by redistributing or exchanging tasks [11, 24]. Strategies using task switching [20, 27, 22] or task exchanges [3, 7] typically transfer tasks between pairs of robots whenever the operation improves the team's performance. The *task swap* mechanism generalizes the idea of pairwise task switches to larger cliques and has begun to be explored recently [16, 29]. In these algorithms each robot has an assigned task at any moment; an important feature is that algorithm may be interrupted at any time. Our recent work [16] showed that strategic choice of task swaps also leads to optimal solutions, thereby bridging decentralized suboptimal task allocation methods and the classical optimal assignment algorithms. Yet, despite its

decentralized nature (lack of any global controller and that only subparts of the system are involved at any moment) the algorithm still depends on the ability to communicate globally.

In this paper, we propose a fully decentralized task allocation algorithm with no stage dependencies that minimizes interactions between robots so that computation occurs only among local cliques. This is achieved by carefully analyzing and redefining the interactions between the search procedures that assess which tasks should be swapped. Although the algorithm 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 step with maximum convergence toward the optimal solution. Since local communication may impose limits that run against global optimality, we first formulate a local optimality property, and then analyze it with duality theory and graph relaxation techniques. Proof of the decentralized nature of the method is proved using group theory.

II. Problem Description and Preliminaries

We consider the multi-robot task assignment problem where the solution is an association of each robot to exactly one task, denoted SR-ST-IA by [9]. 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; dummy robots or tasks can be added otherwise.)

A. Assignment Matrix and Permutation Matrix

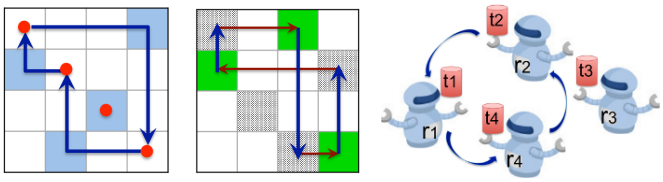
Let binary variable x_{ij} denote the assignment between robot task pair (i, j) so that x_{ij} is equal to 1 if assigned and 0 if unassigned, then an *assignment matrix* can be denoted as $\mathbf{X} = (x_{ij})_{n\times n}$. Since our assignment is a one-to-one mapping, in each row and each column of $\mathbf{X}$ there must be only one entry with value 1 and all others 0s.

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

$$\mathbf{P} = \begin{pmatrix} \mathbf{e}_{k_1} \\ \mathbf{e}_{k_2} \\ \vdots \\ \mathbf{e}_{k_n} \end{pmatrix}, \; k_i \neq k_j \text{ if } i \neq j. \tag{1}$$

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,

$$\begin{aligned} \mathbf{PX} &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathbf{X}'. \end{aligned} \tag{2}$$



(a) (b) (c)

Fig. 1. Illustration for Eq. (2)–(5). (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) 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 $\mathbf{X}' = \mathbf{PX}$. (c) Robots r_1, r_4 and r_2 exchanged their tasks along a closed cycle (142), *e.g.*, r_1 switches to r_4's task after permutation.

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

$$\boldsymbol{\pi} = [\pi(1), \pi(2), \cdots, \pi(n)]^T \tag{3}$$

where $\pi(i)$ is the assigned task for robot i (*i.e.*, the index of elements in $\boldsymbol{\pi}$). Then the example in Eq. (2) becomes

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

Eq. (4) clearly reveals the change of assignment. Comparing $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$, we can observe that the allocated task $\pi(1)$ for robot r_1 (row index 1) is changed from 1 to 3 (*i.e.,* t_1 to t_3), where 3 is $\pi(4)$ in $\boldsymbol{\pi}$. We write this adjustment as $\pi(1) \mapsto \pi(4)$; similarly, for robot r_4, we have $\pi(4) \mapsto \pi(2)$; and for robot r_2, we have $\pi(2) \mapsto \pi(1)$. Robot r_3 keeps task t_2, having no change. For the three robots that have changed tasks, a cycle is formed among them:

$$\pi(1) \mapsto \pi(4) \mapsto \pi(2) \mapsto \pi(1). \tag{5}$$

***Definition** 2.1:* A *Permutation Cycle* with length K is an ordered chain of K distinct elements

$$i_1 \mapsto i_2 \mapsto \cdots \mapsto i_K \mapsto i_1, \tag{6}$$

where i_k denotes the index of the elements. A permutation represents the switching from one element to its successor along the permutation cycle:

$$\pi(i_k) = \pi(i_{(k+1)\%K}), \quad k = 1, \cdots, K. \tag{7}$$

We write the cycle in Eq. (5) as $(142)(3)$ or simply (142) 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** 2.2: 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.*, $(142) = (421) = (214)$ but $(142) \neq (124)$.

B. Assignment Optimization

This classic assignment problem can be formulated with a pair of linear programs. One is a minimization formulation called the *primal program* $\mathcal{P}(R,T)$:

$$\begin{aligned} \text{minimize } & f(R,T) = \sum_{i\in R, j\in T} c_{ij}x_{ij}, \\ \text{subject to } & \sum_{j\in T} x_{ij} = 1, \quad \forall i \in R, \\ & \sum_{i\in R} x_{ij} = 1, \quad \forall j \in T, \\ & x_{ij} \geq 0, \quad \forall i \in R, j \in T. \end{aligned} \tag{8}$$

where each x_{ij} represents a primal variable. The problem turns into an integer problem if it is solved via some combinatorial optimization algorithm and eventually each x_{ij} will equal 0 or 1 in the solution. 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* $\mathcal{D}(R,T)$:

$$\begin{aligned} \text{maximize } & h(R,T) = \sum_{i\in R} u_i + \sum_{j\in T} v_j, \\ \text{subject to } & u_i + v_j \leq c_{ij}, \quad \forall i \in R, j \in T. \end{aligned} \tag{9}$$

***Theorem** 2.1:* (*The Duality Theorem* [5]) If two programs $\mathcal{P}(R,T)$ and $\mathcal{D}(R,T)$ are feasible, then $f(R,T) \geq h(R,T)$. 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 easy, 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 can be derived:

***Theorem** 2.2:* (*The Complementary Slackness Theorem* [5]) 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

$$x_{ij}(c_{ij} - u_i - v_j) = 0, \quad \forall i \in R, j \in T. \tag{10}$$

Remark: Eq. (10) 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** 2.3: Reduced costs* are defined as

$$\bar{c}_{ij} = c_{ij} - u_i - v_j, \quad \forall i \in R, j \in T. \tag{11}$$

The constraint in Program (9) implies that only if $\bar{c}_{ij} \geq 0$ will the robot-task pair (i,j) be *feasible* .

III. Permutation Group and Task Swaps

The formulation of the assignment problem as a matching problem is well known [2]. Unlike those popular treatments, we show that the assignment may also be formulated using group theoretic concepts, which is convenient for understanding and analyzing aspects of its decentralization.

A. Permutation Group

***Definition** 3.1:* 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*): $\exists a' \in \mathcal{G}, a * a' = e = a' * a,\ \forall a \in \mathcal{G}$.

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 operation given by composition, viewing permutations as functions from $R(= T)$ to itself.

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

$$\mathcal{G} = \{g_1, g_2, \cdots, g_m\} \tag{12}$$

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

$$(g_i(g_j(\cdots(g_k\boldsymbol{\pi}))) = (g_i g_j \cdots g_k)\boldsymbol{\pi} = g'\boldsymbol{\pi}, \quad g' \in \mathcal{G}, \tag{13}$$

where multiplicative binary operator $*$ is omitted.

***Proposition** 3.1:* Every permutation in S_n can be written as a product of disjoint cycles. Disjoint cyclic permutations are subject to the commutative law.

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

***Proposition** 3.2:* Any permutation cycle can be written as a product of transpositions (cycles of length 2).

$$(i_1 i_2 \cdots i_k) = (i_1 i_2)(i_2 i_3) \cdots (i_{k-1} i_k) \tag{14}$$

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. With these observations, we have:

***Lemma** 3.1:* 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,

$$\begin{aligned} g &= (i_1 i_2 \cdots i_k) \\ &= (i_1 i_2)\cdots(i_{p-1} i_p)(i_p i_{p+1})\cdots(i_{k-1} i_k) \\ &= \big((i_1 i_2)\cdots(i_{p-1} i_p)\big)\big((i_p i_{p+1})\cdots(i_{k-1} i_k)\big) \\ &= (i_1 i_2 \cdots i_p)(i_p \cdots i_k). \end{aligned} \tag{15}$$

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. (15) do not commute because the element i_p is involved in two resulting non-disjoint cycles, which must be executed 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** 3.1:* Two non-disjoint cycles sharing a common element can be further decomposed into three cycles, among which two cycles become disjoint depending on an appropriate adjustment of the remaining cycle of length 3 (termed *3-cycle*).

Proof: Following the notation in Lemma 3.1, assume we have two cycles with a common element i_p. First, we need to reorder the cycles so that element i_p appears notationally at the end of the cycle, then

$$\begin{aligned} &(i_1 \cdots i_{p-1} i_p)(i_p i_{p+1} \cdots i_k) \\ =&(i_1 \cdots i_{p-1})(i_{p-1} i_p)(i_p i_{p+1})(i_{p+1} \cdots i_k) \\ =&(i_1 \cdots i_{p-1})(i_{p-1} i_p i_{p+1})(i_{p+1} \cdots i_k). \end{aligned} \tag{16}$$

It shows that two cycles become three cycles where the middle one $(i_{p-1} i_p i_{p+1})$ is the 3-cycle. Neither of the other two ending cycles $(i_1 \cdots i_{p-1})$ and $(i_{p+1} \cdots i_k)$ contain i_p and are thus disjoint.

The ending cycles $(i_1 \cdots i_{p-1})$ and $(i_{p+1} \cdots i_k)$ can commute and switch order only if the 3-cycle is adjusted from $(i_{p-1} i_p i_{p+1})$ to $\tilde{g} = (i_{p+1} i_p) g_t g_h (i_p i_{p-1})$, where g_t and g_h denote the operation of swapping two elements at the tail and head in the cycle notation, respectively. More formally, after the adjustment we have

$$\begin{aligned} &(i_{p+1} \cdots i_k)\tilde{g}(i_1 \cdots i_{p-1}) \\ =&(i_{p+2} \cdots i_k i_{p+1})(i_{p+1} i_p) g_t g_h (i_p i_{p-1})(i_{p-1} i_1 \cdots i_{p-2}) \\ =&(i_{p+2} \cdots i_k i_{p+1} i_p) g_t g_h (i_p i_{p-1} i_1 \cdots i_{p-2}) \\ =&(i_{p+2} \cdots i_k i_p i_{p+1})(i_{p-1} i_p i_1 \cdots i_{p-2}) \\ =&(i_p \cdots i_k)(i_1 \cdots i_p), \end{aligned} \tag{17}$$

which switches the operation order of Eq. (16). ■

Conversely, one may regard the 3-cycle 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.

B. Cyclic Permutation vs. Strategic Task Swaps

The task swaps in our previous work [16] 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 more crucially strategic way. Fig. 1(a) shows a permutation cycle whereas the cycle in Fig. 1(b) is a swap loop that involves two types of dark entries. A swap loop can be transformed to a permutation cycle by dropping all even number entries on the loop (such that only the information of robots/rows remains in the cycle notation). Each robot on a swap loop substitutes its task with its successor's along the closed orbit to update the assignment.

The task-swap assignment algorithm is essentially a *primal*-based algorithm [1], which 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)), and the optimal solution exists when the dual also becomes feasible. It is 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 in the centralized context. The main steps are pseudo-coded in Algorithm III.1.

Algorithm III.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. (18)
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. (19), 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

A spanning tree-like data structure in the reduced cost matrix is used to search for swap loops. Searching starts from an entry (i_0, j_0) with infeasible reduced cost ($\bar{c}_{i_0 j_0} < 0$) and new entries are added as tree nodes correspondingly 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

$$\begin{aligned} &\textit{Update } R_v, T_v \textit{ for } i' : \\ &\quad R_v = R_v \cup \{i'\}, \\ &\quad T_v = T_v \cup \{j' \mid \bar{c}_{i'j'} = 0, \forall j' \in T \setminus T_v\}. \end{aligned} \tag{18}$$

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.

$$\begin{aligned} &\textit{Adjust } \mathbf{u}, \mathbf{v}: \\ &\quad \delta = \min \{\bar{c}_{ij} \mid i \in R_v, j \in T \setminus T_v\} \\ &\quad u_i = u_i + \delta, \quad \forall i \in R_v, \\ &\quad v_j = v_j - \delta, \quad \forall j \in T_v. \end{aligned} \tag{19}$$

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

$$\begin{aligned} &f(\mathbf{X}') - f(\mathbf{X}) \\ =&\sum_{i\in R, j\in T} x'_{ij}c'_{ij} - \sum_{i\in R, j\in T} x_{ij}c_{ij} \;=\; \sum_{(i,j)\in L} c'_{ij} - \sum_{(i,j)\in L} c_{ij} \\ =&\sum_{(i,j)\in L} \big(c'_{ij} - (u_i + \delta) - (v_j - \delta)\big) - \sum_{(i,j)\in L} (c_{ij} - u_i - v_j) \\ =&\sum_{(i,j)\in L} \bar{c}'_{ij} - \sum_{(i,j)\in L} \bar{c}_{ij} \;=\; \sum_{(i,j)\in L} \bar{c}'_{ij} \;=\; \bar{c}'_{i_0j_0} \end{aligned} \tag{20}$$

The new solution is improved if $\bar{c}'_{i_0j_0} < 0$, and the overall cost is reduced by an amount of $|\bar{c}'_{i_0j_0}|$ after swapping tasks along the loop.

IV. Decentralized Task Swaps

The preceding formulation has much potential for a decentralized implementations because each stage is likely to involve only a subset of the robots. But some challenges must still be overcome. Firstly, there is the issue of stage dependencies. A robot cannot be involved in two spanning trees simultaneously because the iterative dual updates must proceed sequentially. Secondly, there is the question of robots gaining access to the required information. To reach the optimal solution, the algorithm requires that all robots in the spanning tree to be able to reach one another. Unfortunately this is often an unrealistic assumption in multi-robot systems with limited communication capabilities.

The proposed decentralized method aims at addressing the above problems:

(A.) The swap loop is searched by spanning a tree directly on the network topology rather than the reduced cost matrix so that the dual variables (stage dependence) are eliminated;

(B.) The algorithm searches in a local solution space comprised of only its directly reachable neighbors. Each swap loop maximizes the cost reduction given the local information, which is subject to the network topology.

These aspects are detailed in the following subsections.

A. Spanning Tree on the Transformed Graph

Lemma 4.1: 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

$$c_L = \sum_{(i,j)\in L, x_{ij}=0} \bar{c}_{ij}, \tag{21}$$

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

$$c_L = f(\mathbf{X}') - f(\mathbf{X}). \tag{22}$$

Proof: Proof proceeds from the right side to the left side. From Eq. (20), we have

$$f(\mathbf{X}') - f(\mathbf{X}) = \bar{c}'_{i_0j_0} = c_{i_0j_0} - u'_{i_0} - v'_{j_0}. \tag{23}$$

Since $j_0 \in T_v$ but $i_0 \notin R_v$, Eq. (19) 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,

$$\begin{aligned} &f(\mathbf{X}') - f(\mathbf{X}) = c_{i_0j_0} - u_{i_0} - v_{j'_0} \\ =&c_{i_0j_0} - u_{i_0} - (v_{j_0} - (\delta_1 + \delta_2 + \cdots + \delta_l)) \\ =&\bar{c}_{i_0j_0} + (\delta_1 + \delta_2 + \cdots + \delta_l), \end{aligned} \tag{24}$$

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

$$\begin{aligned} &f(\mathbf{X}') - f(\mathbf{X}) = \bar{c}_{i_0j_0} + c_P \\ =&\bar{c}_{i_0j_0} + \sum_{(i,j)\in L\setminus\{(i_0,j_0)\}, x_{ij}=0} \bar{c}_{ij} \\ =&\sum_{(i,j)\in L, x_{ij}=0} \bar{c}_{ij} \;=\; c_L. \end{aligned} \tag{25}$$

■

Lemma 4.1 reveals that it is only unassigned reduced costs that effect improvements to the solution and, except $\bar{c}_{i_0j_0}$, all other reduced costs on the loop are positive. With this observation, we construct a graph $G = (V, E)$ where we collect each robot and its assigned task 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 immediately leads to the following theorem.

Theorem 4.1: The task swap loop searching problem is transformed to searching for a cycle on a standard directed graph $G = (V, E)$, and the total cycle weight c_L is exactly the cost reduction between new and old assignment solutions.

Fig. 2 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 $\bar{c}_{r_1t_1}$ at the leftmost represent the infeasible entries which actually are not on the graph (they will be used to close the loop). The spanning tree grows by adding new edges with the least weight, and c_P is the path cost between the root and a leaf node on the tree. Finally, an infeasible 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 task swaps.

Theorem 4.2: Spanning tree of Algorithm III.1 is a *greedy* searching method, and the swap loop found by it does not guarantee the maximal reduction in the cost, *i.e.*, the convergence step size might be suboptimal.

*The symbols v, u are reused as vertices and edges in the remainder of the paper. They are not to be confused their use earlier as dual variables.

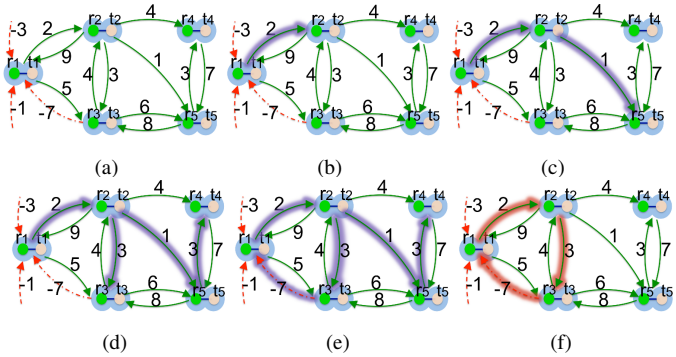


Fig. 2. A swap loop is found on the transformed graph built on the network topology. Solid edges with $w(v_\alpha, v_\beta) = \bar{c}_{r_\alpha \pi(r_\beta)} > 0$ represent the connectivity of the network. The dashed edges connecting to the leftmost node (the starting node) are not actually part of the 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 task swaps, the assigned robot-task pairs in super nodes are changed. The connectivity of graph is also updated.

Proof: In each iteration of Algorithm III.1, only nodes that are connected to the tree but not yet on the tree will be considered, and those nodes (there may be more than one) with minimum connecting edge weight will be added as new leaf nodes. Thus, the tree always exploits the locally optimal choice first and spans in a way analogous to a breath-first-search (BFS). Therefore the searching process is 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. ■

B. Refining Swap Loop via Relaxation

Theorem 4.2 indicates a direction for improvement of the (local) search. Lemma 4.1 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.

Algorithm IV.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: Q_p pops the top node, assuming v; $S = S \cup \{v\}$
6: **for each** node u on an outgoing edge of v **do**
7: **if** $u.d > v.d + w(u, v)$ **then**
8: $u.d = v.d + w(u, v)$, set u's predecessor as v
9: 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 which enrich the searching information. 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 $c_L < 0$ might appear earlier after paths are refined, which reduces the communication load due to an earlier termination.

Additionally, since the relaxation is carried out through comparing 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. This can be regarded as an *anytime* feature since the algorithm can stop at any point in time after a swap loop is detected. Additional relaxation iterations allow the loop to be refined toward better quality. Once all the nodes on the loop are in set S, the swap loop converges to that which reduces the cost the most.

The time complexity for searching for a swap loop is improved over Algorithm III.1. Specifically, given a total number 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 IV.1 requires only $O(|E| + |V|\lg|V|)$, which is $O(n^2)$ in the worse case (analysis is analogous to the Dijkstra's algorithm); in contrast, Algorithm III.1 needs $O(n^2 \lg n)$ to finish the spanning tree starting from the same root [16].

C. Decentralized Algorithm

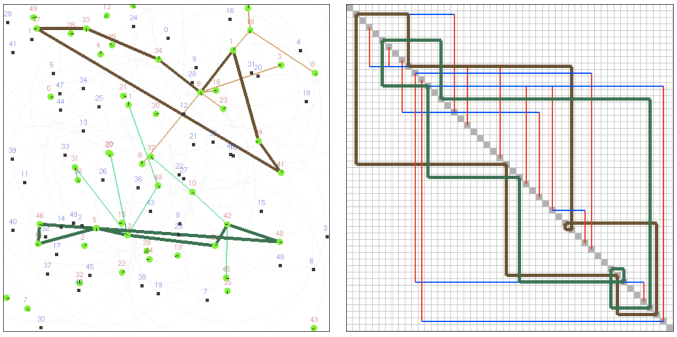
Thus far, the two aforementioned hurdles for decentralization — stage dependence and requirement of global communication — have been eliminated. Consequently, multiple swap loops can be searched concurrently and any robot is allowed to participate in multiple searching processes simultaneously. Robots communicate with neighbors by passing messages containing the latest spanning tree information. 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.

But both Algorithm III.1 and IV.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).

Algorithm IV.2 Decentralized Implementation

1: */* Each robot i maintains a record of neighbors $N(i)$ and their assignment information. */*
2: Select the root: *e.g.*, root i' can be voted among neighbors by $(i', j') = \text{argmin}_{(i,j)} \bar{c}_{ij},\ \forall i \in N(i), j \in T$
3: Each root starts spanning tree on the transformed graph
4: **for each** robot i **do**
5: **if** robot i is root of some tree **then**
6: **if** A (optimal) loop is formed **then**
7: Revisit those in-loop robots to check loop validity
8: **if** loop is valid **then**
9: Notify other robots on the same tree to stop
10: Execute task swaps following the loop
11: Execute assignment of 3-cycles
12: **else**
13: **if** i receives a message **then**
14: Span and relax the tree following Algorithm IV.1
15: **if** the path P_i to i satisfies $\bar{c}_{i'j'} + c_{P_i} > 0$ **then**
16: Stop spanning from this node

Analysis of the decomposition of permutation cycles in Theorem 3.1 shows that non-disjoint cycles can be decom-



(a) (b)

Fig. 3. 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. (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.

posed into two disjoint cycles with a remainder of a 3-cycle. We use this observation to coordinate multiple non-disjoint task swap loops. Assume task swapping along a loop L' : $(\cdots i'_{p-1} i_p i'_{p+1} \cdots)$ is executed first, and a common robot i_p is also involved in the loop formed thereafter L : $(\cdots i_{p-1} i_p i_{p+1} \cdots)$, then the 3-cycle is $(i'_{p-1} i_p i_{p+1})$. We first check the validity of the later loop L by comparing the changes in reduced costs that are associated with the 3-cycle. 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 still valid if and only if

$$\bar{c}_{i_p j'_p} - \bar{c}_{i_p j_p} < c_L, \tag{26}$$

where c_L is the total cost of loop L as defined before.

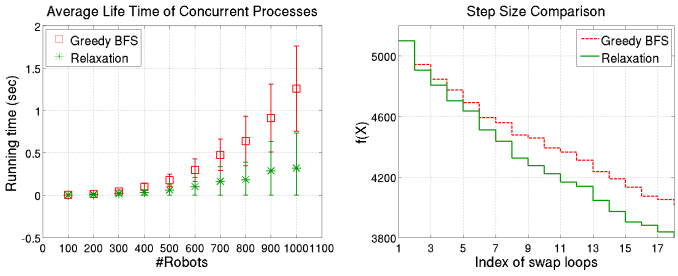
More generally, if two loops share multiple common robots, the changes in reduced costs associated with these common robots are summed up and compared with c_L. Swapping of the tasks in an invalid loop is aborted. Otherwise, if the loop is valid, following Theorem 3.1, an extra assignment can be computed among the robots of the 3-cycle to further refine the assignment (the 3 robots are easily located since they are adjacent in the loop and are neighbors in the network).

In the decentralized implementation, each robot carries out Algorithm IV.1 to maximize the local solution quality. The whole algorithm appears in Algorithm IV.2.

Finally, it is the constraints imposed by the network connectivity which are responsible for any resulting suboptimality:

***Theorem** 4.3:* Algorithm III.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 4.1 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. ■



(a) (b)

Fig. 4. Local searching with optimized steps. With the relaxation described in Algorithm IV.1, (a) the practical running time is improved; (b) the step size is optimized .

V. 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 via a dispatching scenario: a group of robots in the plane must visit a set of destination waypoints. 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. 3 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, 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. 3(a) also shows that in order to find a swap loop, only a subset of robots (and their tasks) need to be involved. Fig. 3(b) shows the corresponding swap loops in the reduced cost matrix.

A. Optimized Local Searching

Local searching with relaxation improves the greedy BFS search described in [16]. Fig. 4(a) shows the practical running time in order to search for a swap loop. We can see that the time required for our 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. 4(b) is a representative example showing that the assignment solution evolves faster with relaxation. The stairs in Fig. 4(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.

B. Solution Convergence

We next compared the convergence performance of the decentralized implementation with our recent task-swap based optimal assignment algorithm [16] 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

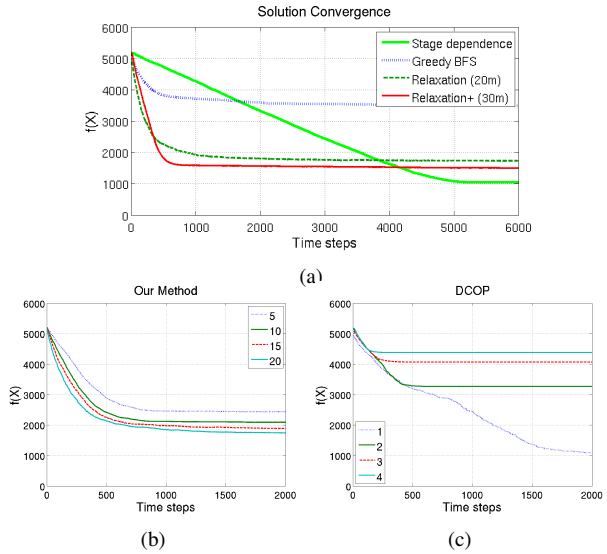


Fig. 5. Solution convergence analysis. (a) The trade-off between the convergence and the optimality for centralized, greedy BFS and relaxation methods of different communication radii (100 robots); (b) Our method: performance under different numbers of searching processes (5–20); (c) DCOP: performance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

broadcast them to multiple receivers in a time step (similar to the communication strategy of MURDOCH architecture [8]).

The trade-off between the optimality and decentralization is revealed in Fig. 5(a), where we observe that the optimal algorithm with stage dependence and global communication has a linear trend toward reaching the optimality, whereas the decentralized approaches with limited communication produced inferior results. We note 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 validates Theorem 4.3.

The number of processes used while searching was also varied in order to investigate its impact. Fig. 5(b) shows that with increased concurrency, convergence is quicker and the solution quality is better. It also shows that the margin of improvement diminishes as the number grows. We opted to use a DCOP algorithm for further comparison: since our method dynamically constructs search trees, the asynchronous distributed constraint optimization (ADOPT) [18] method, which also utilizes a search tree structure, was selected. ADOPT requires that a global search tree is constructed wherein 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 similar to the branching technique in the Branch and Bound method, but in a distributed fashion [18]. Fig. 5(c) shows that DCOP can converge to global optimal solution when the tree is maintained throughout

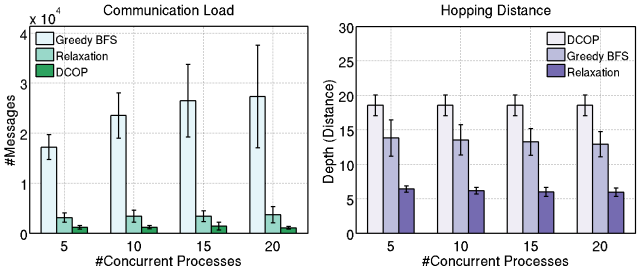


Fig. 6. Communication comparison in terms of (a) the total number of messages and (b) the average longest hopping distance under different number of concurrent processes.

the whole process (*i.e.*, it remains static). However, if the global tree is broken into multiple smaller trees (e.g., owing to the agent failures), then the performance deteriorates drastically. Our method does not suffer from this issue because trees are created and destroyed dynamically and locally.

C. 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. 6(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. 6(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.

VI. 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 transformed 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.

REFERENCES

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

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

[3] Luiz Chaimowicz, Mario F. M. Campos, and Vijay Kumar. Dynamic role assignment for cooperative robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 293–298, 2002.

[4] Anton Chechetka and Katia 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.

[5] George Dantzig. *Linear Programming and Extensions*. Princeton University Press, August 1963.

[6] 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.

[7] 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.

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

[9] Brian P. Gerkey and Maja 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.

[10] Stefano Giordani, Marin Lujak, and Francesco Martinelli. A Distributed Algorithm for the Multi-Robot Task Allocation Problem. *LNCS: Trends in Applied Intelligent Systems*, 6096: 721–730, 2010.

[11] 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.

[12] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

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

[14] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, and Sonal Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems*, 2005.

[15] Lantao Liu and Dylan A. Shell. Multi-robot formation morphing through matching graph. In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2012.

[16] Lantao Liu and Dylan A. Shell. A distributable and computation-flexible assignment algorithm: From local task swapping to global optimality. In *Proceedings of Robotics: Science and Systems*, 2012.

[17] Nathan Michael, Michael M. Zavlanos, Vijay Kumar, and George J. Pappas. Distributed multi-robot task assignment and formation control. In *IEEE Intl. Conf on Robotics and Automation*, pages 128–133, 2008.

[18] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2006.

[19] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 266–271, 2005.

[20] Sanem Sariel and Tucker Balch. A distributed multi-robot cooperation framework for real time task achievement. In *Proceedings of Distributed Autonomous Robotic Systems*, 2006.

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

[22] Cynthia Sung, Nora Ayanian, and Daniela Rus. Improving the performance of multi-robot systems by task switching. In *IEEE International Conference on Robotics and Automation*, pages 2984 – 2991, 2013.

[23] 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.

[24] L. Thomas, A. Rachid, and L. Simon. A distributed tasks allocation scheme in multi-UAV context. In *Proc. ICRA*, pages 3622–3627, 2004.

[25] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of aerial robots. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[26] Matthew Turpin, Nathan Michael, and Vijay Kumar. CAPT: Concurrent assignment and planning of trajectories for multiple robots. *International Journal of Robotics Research*, 33(1):98–112, 2014.

[27] Jens Wawerla and Richard 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.

[28] 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.

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