Robotics: Science and Systems 2012
Sydney, NSW, Australia, July 09-13, 2012

A Distributable and Computation-flexible Assignment Algorithm: From Local Task Swapping to Global Optimality

Lantao Liu
Dept. of Computer Science and Engineering
Texas A&M University
College Station, USA
Email: lantao@cse.tamu.edu

Dylan A. Shell
Dept. of Computer Science and Engineering
Texas A&M University
College Station, USA
Email: dshell@cse.tamu.edu

***Abstract*— The assignment problem arises in multi-robot task-allocation scenarios. This paper introduces an algorithm for solving the assignment problem with several appealing features for online, distributed robotics applications. The method can start with any initial matching and incrementally improve the solution to reach the global optimum, producing valid assignments at any intermediate point. It is an any-time algorithm with an attractive performance profile (quality improves linearly) that, additionally, is comparatively straightforward to implement and is efficient both theoretically ($O(n^3 \lg n)$ complexity is better than widely used solvers) and practically (comparable to the fastest implementation, for up to hundreds of robots/tasks). We present a centralized version and two decentralized variants that trade between computational and communication complexity.**

Inspired by techniques that employ task exchanges between robots, our algorithm guarantees global optimality while using generalized "swap" primitives. The centralized version turns out to be a computational improvement and reinterpretation of the little-known method of Balinski-Gomory, proposed half a century ago. Deeper understanding of the relationship between approximate swap-based techniques —developed by roboticists— and combinatorial optimization techniques, e.g., the Hungarian and Auction algorithms —developed by operations researchers but used extensively by roboticists— is uncovered.

I. INTRODUCTION

A common class of multi-robot task-allocation mechanisms involve estimating the expected cost for each robot's performance of each available task, and matching robots to tasks in order to minimize overall cost. By allocating robots to tasks repeatedly, a team can adapt as circumstances change and demonstrate fluid coordination. A natural tension exists between two factors: running-time is important as it determines how dynamic the team can be, while quality of the allocation reflects the resultant total cost and hence the performance of the team. While the importance of solutions that trade the quality of results against the cost of computation has been established for some time (*e.g.*, review in [1]), the assignment problem underlying efficient task-allocation has received little attention in this regard.

This paper introduces an algorithm that yields a feasible allocation at any point in its execution and an optimal assignment when it runs to completion. The results give an easily characterizable relationship between running time and allocation quality, allowing one factor to be traded for the other, and even for the marginal value of computation to be estimated. Additionally, the algorithm may start from any initial matching so it can be easily used to refine sub-optimal assignments computed by other methods.

But the flexibility afforded by an any-time algorithm will be counterproductive if it comes at too high a cost. The method we describe has strongly polynomial running time and we show that it can be competitive with the fastest existing implementation even for hundreds of robots and tasks. Additionally, the cost can be borne by multiple robots because variants of the algorithm can be executed in a decentralized way. We are unaware of another solution to the assignment problem with these features.

II. RELATED WORK

Task allocation is one of the fundamental problems in distributed multi-robot coordination [2]. Instantaneously assigning individual robots to individual tasks involves solution of the linear-sum assignment problem. This paper draws a connection between methods for (*A.*) improving local performance, *e.g.*, via incremental clique preferences improvement, and (*B.*) allocation methods which seek to solve (or approximate) the global optimum of the assignment.

A. Local Task Exchanges in Task-Allocation

Several researchers have proposed opportunistic methods in which pairs of robots within communication range adjust their workload by redistributing or exchanging tasks between themselves [3, 4, 5], also called O-contracts [6]. These intuitively appealing methods allow for a form of localized, lightweight coordination of the flavor advocated by [7]. Zheng and Koenig [8] recently explored a generalization of the idea in which an exchange mechanism involving K robots (called K-swaps) improves solution quality. They theoretically analyzed and illustrated properties of the method empirically. This paper gives new insight into how generalized swap-like mechanisms can ensure optimality, in our case through something analogous to automatic computation of the necessary value of K. Also, we have characterized the running-time of our method.

B. Optimal Assignment in Task-Allocation

The first and best-known optimal assignment method is Kuhn's $O(n^3)$ Hungarian algorithm [9]. It is a *dual-based* (or generally *primal-dual*) algorithm because the variables in the dual program are maintained as feasible during each iteration in which a primal solution is sought. Many other

assignment algorithms have been developed subsequently (see review [10]) however most are dual-based methods including: augmenting path [11], the auction [12], pseudo-flow [13] algorithms, *etc.* These (and approximations to them) underlie many examples of robot task-allocation, *e.g.,* see [14, 15, 16]. Special mention must be made of market-based methods (*e.g.,* [17, 18]) as they have proliferated presumably on the basis of inspiration from real markets and their naturally distributed operation, and Bertsekas's economic interpretation of dual variables as prices [12]. Fully distributing such methods sacrifices optimality: [19] gives bounds for some auction strategies.

Little work reports using *primal* approaches for task-allocation; researchers who solve the (relaxed) Linear Program directly likely use the popular (and generally non-polynomial time) simplex method [20]. The primal assignment algorithm proposed by Balinski and Gomory [21] is an obscure method that appears entirely unknown within robotics. The relationship is not obvious from their presentation, but their chaining sequence of alternating primal variables is akin to the swap loop transformation we have identified. Our centralized algorithm improves on their run-time performance (they require $O(n^4)$ time). Also, the data structures we employ differ as they were selected to reduce communication cost in the decentralized versions, which is not something they consider.

III. PROBLEM DESCRIPTION & PRELIMINARIES

We consider the multi-robot task assignment problem in which the solution is an association of each robot to exactly one task , denoted SR-ST-IA by [14]. An assignment $\mathcal{A} = \langle R, T \rangle$ consists a set of robots R and a set of tasks T. Given matrix $C = (c_{ij})_{n \times n}$, where $c_{ij} : R \times T \rightarrow \mathbb{R}^+$ represents the cost of having robot i perform task j. In our work, $n = |R| = |T|$, the number of robots is identical to the number of tasks (otherwise dummy rows/columns can be inserted).

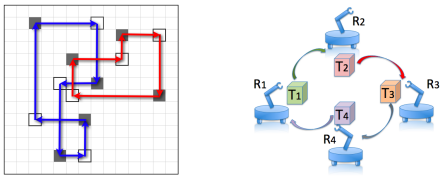
A. Formulations

This problem can be formulated with an equivalent pair of linear programs. The *primal* is a minimization formulation:

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

where an optimal solution eventually is an extreme point of its feasible set (x_{ij} equals to 0 or 1). Let binary matrix $\mathbf{X} = \{x_{ij}\}$, $\forall (i,j)$ contain the primal variables. The constraints $\sum_j x_{ij} = 1$ and $\sum_i x_{ij} = 1$ enforce a *mutual exclusion* property. There are corresponding *dual* vectors $\mathbf{u} = \{u_i\}$ and $\mathbf{v} = \{v_j\}$, with dual linear program:

$$\begin{aligned} \text{maximize } & g = \sum_i u_i + \sum_j v_j, \\ \text{subject to } & u_i + v_j \leq c_{ij}, \quad \forall (i,j). \end{aligned} \tag{2}$$



(a) (b)

Fig. 1. Primal transformations are task swaps. (a) A cost matrix with two independent swap loops, where the shaded and bold-edged squares represent the old and new assigned entries, respectively; (b) Task swapping from an independent swap loop (*e.g.*, blue loop in (a)) among four robots and tasks.

B. Complementary Slackness, Reduced Cost, and Feasibility

The *duality theorems* show that a pair of feasible primal and dual solutions are optimal iff the following is satisfied:

$$x_{ij}(c_{ij} - u_i - v_j) = 0, \qquad \forall (i,j). \tag{3}$$

This *complementary slackness* equation reveals the orthogonal property between the primal and dual variables. The values

$$\bar{c}_{ij} = c_{ij} - u_i - v_j, \qquad \forall (i,j), \tag{4}$$

are called the *reduced costs*. For a maximization dual as shown in Program (2), its constraint shows that an assignment pair (i,j) is *feasible* when and only when $\bar{c}_{ij} \geq 0$.

C. Transformations and Admissibilities

Primal and dual transformations and, in particular, their admissibilities are used later in the paper.

- *Admissible Primal Transformation:* Map $Z_p : \mathbf{X} \mapsto \mathbf{X}'$ is an *admissible primal transformation* if the primal solution quality is better after the transformation, *i.e.* $\mathbf{X}' = Z_p(\mathbf{X})$ is admissible iff $f(\mathbf{X}') < f(\mathbf{X})$ for a minimization problem.
- *Admissible Dual Transformation:* $Z_d : (\mathbf{u}, \mathbf{v}) \mapsto (\mathbf{u}', \mathbf{v}')$ is an *admissible dual transformation* if the size for the set of feasible reduced costs increases, *i.e.*, $(\mathbf{u}', \mathbf{v}') = Z_d(\mathbf{u}, \mathbf{v})$ is admissible iff $|\{(i,j) \mid \bar{c}'_{ij} \geq 0\}| > |\{(i,j) \mid \bar{c}_{ij} \geq 0\}|$.

IV. TASK SWAPPING AND OPTIMALITY

Any primal transformation $\mathbf{X}' = Z_p(\mathbf{X})$ is easily visualized by superimposing both $\mathbf{X}$ and $\mathbf{X}'$ on an assignment matrix. Shown as shaded and bold-edged entries in Fig. 1(a), the transformations can be interpreted as row-wise and column-wise aligned arcs. Connecting the beginning to the end closes the path to form what we call a *swap loop*, which is easily imagined as a subset of robots handing over tasks in a chain, as illustrated in Fig. 1(b).

If a swap loop shares no path segment with any other, it is termed *independent*.

***Theorem** 4.1:* A primal transformation $\mathbf{X}' = Z_p(\mathbf{X})$ where $(\mathbf{X} \neq \mathbf{X}')$ forms a (non-empty) set of independent swap loops.

Proof: The mutual exclusion property proves both parts. Independence: if a path is not independent, there must be at least one segment that is shared by multiple paths. Any such segment contradicts the mutual exclusion constraints since either $\sum_i x_{ij} > 1$, or $\sum_j x'_{ij} > 1$, or both.
Closeness: a non-closed path has end entries that are exposed; but this leads to $\sum_j x'_{ij} = 0$ or $\sum_i x'_{ij} = 0$. ■

Assume $S_{swp} = \{swp_\lambda\}$ $(\lambda \in [1, m])$ is a set of swap loops where swp_λ denotes the λ^{th} swap loop. Let primal

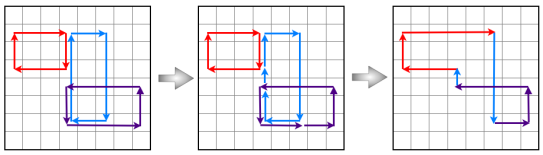


Fig. 2. Amalgamation allows synthesis of complex swap loops from multiple dependent swap loops. Overlapped path segments cancel each other out.

transformation $\mathbf{X} \rightarrow \mathbf{X}^{'}$ with specific set of swap loops S_{swp} also be denoted as $\mathbf{X}^{'} = Z_p^{S_{swp}}(\mathbf{X})$.

***Theorem** 4.2:* A primal transformation involving mutually independent swap loops $S_{swp} = \{swp_1, swp_2, \cdots, swp_m\}$ can be separated and chained in any random order. *i.e.*, $\mathbf{X}^{'} = Z_p^{\{swp_1\}}(Z_p^{\{swp_2\}} \cdots Z_p^{\{swp_m\}}(\mathbf{X}))$.

Proof: A primal transformation is isomorphic to a set of row and column permutations. Assume the row and column permutation matrices (each is a square orthogonal binary doubly stochastic matrix) corresponding to set S_{swp} are $\mathbf{P}$ and $\mathbf{Q}$, so that $\mathbf{PXQ}$ permutes the rows and columns of $\mathbf{X}$ appropriately. If row i is unaffected the i^{th} column of $\mathbf{P}$, $\mathbf{p}_i = \mathbf{e}_i$ (the i^{th} column of the identity matrix) and then $\mathbf{P} = \prod_{\lambda=1}^{m} \mathbf{P}_\lambda$, where $\mathbf{P}_\lambda$ represents the separated permutation matrix for the λ^{th} swap loop, will have a non-interfering form so that the order of product does not matter. Thus we have $\mathbf{X}' = \mathbf{PXQ} = \mathbf{P_1 P_2} \cdots \mathbf{P_m X Q_m Q_{m-1}} \cdots \mathbf{Q_1}$ (order of $\mathbf{P}_\lambda$s do not matter, nor do $\mathbf{Q}_\lambda$s analogously), which is equivalent to $\mathbf{X}' = Z_p^{\{swp_1\}}(Z_p^{\{swp_2\}} \cdots Z_p^{\{swp_m\}}(\mathbf{X}))$. ■

However, many times independent swap loops can not be directly obtained. Instead, an independent swap loop may be composed of multiple *dependent* swap loops that share rows/columns on some path segments.

***Theorem** 4.3:* Two dependent swap loops with overlapping, reversed segments can be amalgamated into a new swap loop, and vise versa.

Proof: A directed path segment can be conveniently represented as vector $\vec{\pi}$. Path segments $\vec{\pi_1}$ and $\vec{\pi_2}$ sharing the same rows or columns, but with different directions, cancel via $\vec{\pi}^{'} = \vec{\pi_1} + \vec{\pi_2}$, which has interpretation as a task (robot) handed from one robot (task) to another, but then passed back again. Such cancellation must form a loop because each merger collapses one pair of such segments, consistently connecting two partial loops. The opposite operation (decomposition) involves analogous reasoning. ■

While ordering of independent swap loops is unimportant, the number, size and, order of dependant loops matter.

***Theorem** 4.4:* When $K < n$, K-swaps are susceptible to local minima.

Proof: A K-swap loop involves at most K robots and K assigned tasks. Quiescence results by reaching equilibrium after sufficient K-swaps so that no more swaps can be executed. Robots and their assigned tasks involved in the K-swap can form a smaller sub-assignment of size K. Thus, we have $\binom{n}{K}$ possible such sub-assignments, and all of them are optimal at equilibrium. Assume the set of these sub-assignments is $S_{\mathcal{A}} = \{\mathcal{A}_\gamma\}$, where $\gamma \in [1, \binom{n}{K}]$. $\mathcal{A}_\gamma = \{R_\gamma, T_\gamma\}$ represents the sub-assignment with robot (task) index set $|R_\gamma| = K$ ($|T_\gamma| = K$). Therefore, the dual program for each sub-assignment is:

$$\max\ g(\mathcal{A}_\gamma) = \sum_{i \in R_\gamma} u_i + \sum_{j \in T_\gamma} v_j, \tag{5}$$

$$\text{subject to}\ \ u_i + v_j \leq c_{ij}, \quad \forall i \in R_\gamma, j \in T_\gamma. \tag{6}$$

If we put all the sub-assignments together, the whole assignment problem can be written in the form

$$\binom{n-1}{K-1}^{-1} \sum_{\gamma \in [1, |S_{\mathcal{A}}|]} \max\ g(\mathcal{A}_\gamma) \tag{7}$$

$$\text{subject to}\ \ u_i + v_j \leq c_{ij} \qquad \forall i, j, \tag{8}$$

where the first term in the product accounts for the repeated summation of each dual variable. By the fact that $\sum_{i \in \mathcal{I}, z_i \in \mathcal{Z}} (\max\ z_i) \geq \max\ \sum_{i \in \mathcal{I}, z_i \in \mathcal{Z}} z_i$, we have

$$\binom{n-1}{K-1}^{-1} \sum_{\gamma \in [1, |S_{\mathcal{A}}|]} \max\ g(\mathcal{A}_\gamma) \geq \max\ g(\mathcal{A}) \tag{9}$$

where $\mathcal{A}$ is the original $n \times n$ assignment. With the duality theorems, this is equivalent to

$$\binom{n-1}{K-1}^{-1} \sum_{\gamma \in [1, |S_{\mathcal{A}}|]} \min\ f(\mathcal{A}_\gamma) \geq \min\ f(\mathcal{A}). \tag{10}$$

So even completing every possible $K-$swap, and doing so until equilibrium is reached, may still end sub-optimally. ■

V. AN OPTIMAL SWAP-BASED PRIMAL METHOD

The preceding results suggest that to obtain the optimal primal transformation, one seeks a set of independent swap loops, but that these can be equivalently sought as a series of dependent swap loops. The primal assignment method we describe achieves it iteratively and avoids local minima because later swaps may correct earlier ones based on "enlarged" views that examine increasing numbers of rows and columns. The essence of the primal assignment method is that, at any time, the primal solution's feasibility is maintained (*i.e.,* the mutual exclusion property is satisfied), while infeasible dual variables are manipulated under the complementary slackness condition. At each iteration either an admissible primal transformation is found, or a new improved set of dual variables are obtained. Once all reduced costs are feasible, the primal and dual solutions simultaneously reach their (equal valued) optimum.

The method is described in Algorithms V.1–V.4 in some detail to ensure that the pseudo-code is appropriate for straightforward implementation.

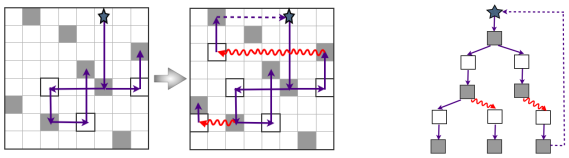
Algorithm V.1 PRE-PROCESS ($\bar{\mathbf{C}}$, $\mathbf{u}$, $\mathbf{v}$)

```
1: initiate n min-heaps h[n] := {null}
2: for i := 1 to n do
3:    for j := 1 to n do
4:       if C̄[i][j] ≥ 0 AND j ≠ a(i) then
5:          make pair p := ⟨label = j, value = C̄[i][j]⟩
6:          insert p into h[i]
7: return min-heaps h
```

Note: Variable u_i and $u[i]$ are equivalent, vector $\mathbf{v} \equiv v[n]$, matrix $\bar{\mathbf{C}} \equiv \bar{C}[n][n]$.

A. Algorithm V.1: Pre-processing

At each stage, the reduced cost matrix $\bar{\mathbf{C}}$ is pre-processed before searching for a swap loop: a separate min-heap is used to maintain the feasible reduced costs in each row, such that smallest values (root elements) can be *extracted* or *removed* efficiently.



(a) (b)

Fig. 3. (a) Path segments are bridged with one another while searching for swap loops. 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) The associated tree data structure that aids efficient searching.

B. Algorithm V.2: Searching for Swap Loops

Any swap loop yields an admissible primal transformation. Loops are sought by bridging path segments in the reduced costs matrix. A horizontal path segment is built from a currently assigned entry to a new entry with reduced cost of zero in the same row. Vertical path segments are implicitly identified as from unassigned entries equal to zeros to the unique assigned entries in the respective column. Fig. 3(a) shows the process. The search uses a tree, expanded in a breadth first fashion, to find the shortest loop; a dead-end (*i.e.*, empty queue) triggers the dual adjustment step.

Algorithm V.2 SWAP_LOOP ($\bar{\mathbf{C}}$, $\mathbf{h}$, $\boldsymbol{\delta}$, x, y)

starting row $r_s := x$, column $t_s := y$, $S_R := S_T := \varnothing$
initiate $r := a^{-1}(y)$, $t := y$, $Vpath(t : r_s \to r)$
push r into queue Q, $color(S_R \cup \{r\}; S_T \cup \{t_s\})$
while Q not empty AND $Q.front \neq r_s$ **do**
  $r := Q.front$, $Q.pop$ once
  initiate set $S_\delta := \{r\}$
  for each $r \in S_\delta$ **do**
    $t := h[r].extract.label$
    while $p(r,t)^\dagger = 0$ **do**
      if $t \notin S_T$ **then**
        $Hpath(r : a(r) \to t)$, $Vpath(t : r \to a^{-1}(t))$
        push $a^{-1}(t)$ into Q, $color(S_R \cup \{a^{-1}(t)\}; S_T \cup \{t\})$
      $h[r].remove$ root element and update root
      update $t := h[r].extract.label$
  if Q empty **then**
    DUAL_ADJ ($\bar{\mathbf{C}}$, Q, $\mathbf{h}$, $\boldsymbol{\delta}$, S_δ, r_s, t_s)
    if updated S_δ not empty **then**
      go to STEP 7
    return
$Hpath(r_s : t \to t_s)$, form a loop

†: $p(\cdot)$ is a projection of reduced cost, defined in (12) on page 5.

In Algorithm V.2, function $t = a(r)$ denotes the assignent for r is t and thus is used to extract the column index with a given row index; the inverse does the reverse. Horizontal (vertical) segments are constructed via $Hpath(cur\_row : col1 \to col2)$ ($Vpath(cur\_col : row1 \to row2)$), where the three domains represent the current row (column) containing the path, the starting column (row) and the ending column (row) for the segment, respectively. The *coloring* on visited rows/columns is merely the set union operation.

C. Algorithm V.3: Dual Adjustments

Dual adjustment introduces entries with reduced costs equal to zero so that the tree can be expanded. This is done by changing the values of dual variables, which indirectly changes the reduced costs of corresponding entries. Doing so can only increase the size of the set of feasible reduced costs, thus the dual adjustment will never deteriorate the current result. The method subtracts the smallest feasible reduced cost from all visited (colored) rows and adds it to every visited columns, producing at least one new 0-valued reduce cost(s). Red arrows in Fig. 3 illustrate such procedure.

Algorithm V.3 DUAL_ADJ ($\bar{\mathbf{C}}$, Q, $\mathbf{h}$, $\boldsymbol{\delta}$, S_δ, r_s, t_s)

array $top\_d[n] := \{\infty\}$, $col\_d[n] := \{0\}$
for $i := 1$ to n **do**
  if row $i \in S_R$ **then**
    $top\_d[i] := p(i, h[i].extract.label)$
$min\_\delta := \min\{top\_d[i]\}$
if $min\_\delta > 0$ **then**
  update $S_\delta := \{i \mid top\_d[i] = min\_\delta\}$
else
  $min\_\delta := -p(r_s, t_s)$
for $i := 1$ to n **do**
  if row $i \in S_R$ **then**
    update $\delta[a(i)] := \delta[a(i)] + min\_\delta$
  $col\_d[i] := p(i, t_s)$
if $\min\{col\_d[i]\} \geq 0$ **then**
  terminate current stage
update starting row $r_s := \text{argmin}_i\{col\_d[i]\}$
if $r_s \in S_R$ **then**
  $Hpath(r_s : a(r_s) \to t_s)$, form a loop
  terminate current swap loop searching

The whole algorithm is organized in Algorithm V.4.

Algorithm V.4 PRIMAL ($\mathbf{C}$)

init arrays $u[n] := \{0\}$, $v[n] := \text{diag}(\mathbf{C})$, $\bar{C}[n][n] := \{0\}$
for $i := 1$ to n **do**
  update matrix $\bar{\mathbf{C}}$ with $\bar{c}_{i'j'} = c_{i'j'} - u_{i'} - v_{j'}$, $\forall i', j'$
  if $\min\{\bar{C}[:][i]\} < 0$ **then**
    array $\delta[n] := \{0\}$
    heap $h[n] :=$ PRE-PROCESS($\bar{\mathbf{C}}$, $\mathbf{u}$, $\mathbf{v}$)
    check the ith column of $\bar{\mathbf{C}}$, get smallest-valued entry (x, y)
    SWAP_LOOP($\bar{\mathbf{C}}$, $\boldsymbol{\delta}$, $\mathbf{h}$, x, y)
    for $j := 1$ to n **do**
      $u[j] := u[j] + \delta(a(j))$, $v[j] := v[j] - \delta[j]$
    $v[y] := v[y] - |C[x][y] - u_x - v_y|$ so that $\bar{C}[x][y] = 0$
    if a swap loop found, swap tasks to augment solution

Next, we return to the relation of this method to the Balinski-Gomory's primal technique [21]. Theoretical complexity and empirical results below show the superiority of the swap-based approach. Nevertheless, it is worthwhile to address the conceptual differences in detail as a common underlying idea is involved: they employ an iterative labelling and updating techniques to seek a chaining sequence of alternating primal variables, which are used to adjust and augment the primal solutions. Three aspects worth highlighting are:

1) The swap loop search incorporates the dual adjustment procedure. Balinski-Gomory's method may require n rounds of traversals and cost $O(n)$ times more than the traversal based on building and maintaining our tree. This modification is most significant for the decentralized context as each traversal involves communication overhead.
2) Instead of directly updating $\mathbf{u}, \mathbf{v}$, the δ array accumulates the dual variable adjustments during each stage. All updates

are transferred to $\mathbf{u}$ and $\mathbf{v}$ after the whole stage is terminated:

$$u_i' = \begin{cases} u_i + \sum_\omega \delta_{a(i)}^{(\omega)}, & \forall i \in S_R \\ u_i \text{ otherwise} \end{cases} \quad v_j' = \begin{cases} v_j - \sum_\omega \delta_j^{(\omega)}, & \forall j \in S_T \\ v_j \text{ otherwise} \end{cases} \tag{11}$$

where S_R and S_T are index sets of colored rows and columns, respectively, and ω is the index of iterations. The benefit lies in that reduced costs in the whole matrix need not updated on each dual variables adjustment. Instead, query of reduced cost $\bar{c}_{ij}'$ for individual entry (i, j) during an intermediate stage can be obtained via a projection $p(i, j)$:

$$\bar{c}_{ij}' = p(i, j) = \bar{c}_{ij} + \delta_j - \delta_{a(i)}. \tag{12}$$

3) Swap loops are found more efficiently: for example, the heaps, coloring sets and tree with alternating tree nodes — assigned entries with n-ary branches, and unassigned entries with unary branches — quickly track the formation of loops even when the root is modified (Step 16 of Algorithm V.3).

D. Correctness

Assume the starting infeasible entry of matrix is (k, l) with reduced cost $\bar{c}_{kl} = c_{kl} - u_k - v_l < 0$.

***Theorem** 5.1:* Once a task swap loop starting from entry (k, l) is obtained, the task swaps must lead to an admissible primal transformation.

Proof: Term $c_{ij}x_{ij}$ contributes to $f(\mathbf{X}) = \sum_{i,j} c_{ij}x_{ij}$, only when binary variable $x_{ij} = 1$. Also $c_{ij} = u_i + v_j$ via (3). From (11):

$$\begin{aligned} f(\mathbf{X}') - f(\mathbf{X}) &= \sum_{i,j}(u_i' + v_j') - \sum_{i,j}(u_i + v_j) \\ &= \sum_i \left(u_i' - u_i + v_{a(i)}' - v_{a(i)}\right) \\ &= \sum_i \left(\sum_\omega \delta_{a(i)}^{(\omega)} - \sum_\omega \delta_{a(i)}^{(\omega)}\right) + \xi = \xi, \end{aligned} \tag{13}$$

where $\xi = v_l' - v_l = -|c_{kl} - u_k - v_l| < 0$ (see Step 11 in Algorithm V.4). So after a swap, the value of the primal objective must decrease. ■

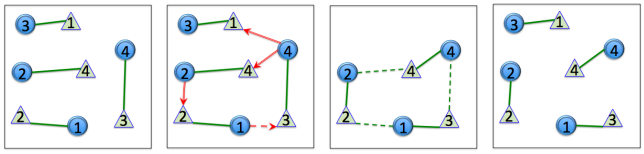
***Theorem** 5.2:* If no task swap loop starting from entry (k, l) is found, an admissible dual transformation must be produced.

Proof: First, feasible reduced costs remain feasible:

$$\begin{aligned} \bar{c}_{ij}' &= c_{ij} - u_i' - v_j' \\ &= c_{ij} - u_i - \sum_\omega \delta_{a(i)}^{(\omega)} - v_j + \sum_\omega \delta_j^{(\omega)} \\ &= \begin{cases} \bar{c}_{ij} \geq 0, & \forall i \in S_R, j \in S_T \\ \bar{c}_{ij} - \sum_\omega \delta_{a(i)}^{(\omega)} \geq 0, & \forall i \in S_R, j \notin S_T. \end{cases} \end{aligned} \tag{14}$$

Second, at least $\bar{c}_{kl}$ will become feasible which leads to the termination before formation of a swap loop, even in the sophisticated strategy allowing dynamic updating of starting entry (See step 16 of Algorithm V.3). This proves that the set of feasible reduced costs must increase. ■

Theorems 5.1 and 5.2 also imply that an admissible primal transformation must be an admissible dual transformation, but not vice versa. So a set of feasible reduced costs must increase over stages that start from infeasible entries, proving that the algorithm must terminate. Algorithm V.4 requires at most n stages because in each stage the smallest infeasible reduced cost in each column is selected (Step 7 of Algorithm V.4), all other infeasible entries in the same column will thus, also become feasible.



(a) (b) (c) (d)

Fig. 4. Swap loop searching in a multi-robot system using Euclidean distance as cost metric. Circles represent robots and triangles denote the tasks. The graphs can also be interpreted as Hypergraphs.

E. Time Complexity

The pre-processing using min-heaps for any stage requires $O(n^2 \lg n)$. During each stage, there are at most n DUAL_ADJs for the worst case and each needs $O(n)$ time to obtain min_δ via the heaps. Visited columns are colored in a sorted set and are never considered for future path bridging in any given stage. There are at most n^2 entries to color and check, each costs $O(\lg n)$, yielding $O(n^2 \lg n)$ per stage. Therefore, the total time complexity for the whole algorithm is $O(n^3 \lg n)$ and the light-weight operations lead to a small constant factor.

By way of comparison, Balinski-Gomory's primal method [21] uses $O(n^2)$ searching steps with $O(n^2)$ time complexity for each step. Some researchers [22, 23] have suggested that it may possible to further improve the time complexity to $O(n^3)$ using techniques such as the blossom method [11]. To the best of our knowledge, no such variant has been forthcoming.

In addition, although min-heaps in Algorithm V.1 are created in a separate step for the algorithmic description reason, in practice they can be constructed on the fly only when they are required, through which a better practical running time can be obtained although the time complexity is unchanged. Experimental results also show that using a fast approximation algorithm for initialization produces running times close to the fastest existing assignment algorithms with $O(n^3)$ time complexity.

VI. DISTRIBUTED VARIANTS

Distributed variants of our primal method are easily obtained. Swap loops are searched via message passing: messages carrying dual variables $(\mathbf{u}, \mathbf{v})$ and dual updates $\boldsymbol{\delta}$ are passed down the tree while searching progresses. The idea is illustrated in Fig. 4 for a single swap loop searching stage with four robots. The green lines show the initial pairwise robot-task assignment; the red arrows show bridging edges found by searching for a swap loop starting from a selected pair. If the path ending pair connects to the starting pair, then a swap loop has been found (Fig 4(c)) and tasks may be exchanged among robots in the loop. The new assignment is finally shown in Fig. 4(d).

Unlike centralized algorithms, the cost matrix may be not globally visible. Instead, each robot maintains and manipulates its own cost vector associated with all tasks. A noteworthy feature is that a robot need not know the cost information of other robots, since the two arrays of dual variables are

shared. We do assume that the initial assignment solution and the corresponding costs for the assigned robot-task pairs are known by all robots, so the initial reduced costs for each robot may be calculated locally.

The algorithm has two roles: an *organizer* robot that holds the starting infeasible entry, and the remainder being *member* robots (but with unique IDs). The organizer initiates a swap loop search iteration at stage m, by communicating a message containing the dual information $\mathbf{u}_{m-1}, \mathbf{v}_{m-1}$ obtained from stage $m-1$, as well as a newly created dual increment vector $\boldsymbol{\delta}_m$. A successor robot is located from either the assignment information or the newly found feasible and orthogonal entries satisfying the complementary slackness, as presented in the centralized version. When a path can no longer be expanded, member robots at the respective "dead-ends" request a dual adjustment from the organizer. Once the organizer has collected requests equal to the number of branches, it computes and transmits $\boldsymbol{\delta}_m$. The process continues until a swap loop is found and tasks are exchanged. At this point, the organizer either re-elects itself as next stage's organizer, or hands over the role to other robots, based on different strategies discussed below. The roles are described in Algorithms VI.1 and VI.2.

Algorithm VI.1 Organizer ($\mathbf{u}_{m-1}$, $\mathbf{v}_{m-1}$, $\boldsymbol{\delta}_m$)

initiate: ▷ only once
  decide starting entry x_m, y_m for current stage m
  send $msg(\mathbf{u}_{m-1}, \mathbf{v}_{m-1})$ to member with ID $a^{-1}(y)$
listening:
  if all involved IDs request dual adjustments **then**
    compute $\boldsymbol{\delta}_m$, send it it to corresponding ID(s)
  endif
if swap loop formed **then**
  with $\boldsymbol{\delta}_m$, update $\mathbf{u}_{m-1}, \mathbf{v}_{m-1}$ to $\mathbf{u}_m, \mathbf{v}_m$ for next stage
  decide next organizer j and send $msg(\mathbf{u}_m, \mathbf{v}_m)$ to ID j

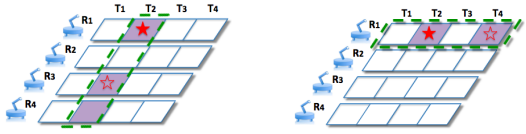
Algorithm VI.2 Member[i] (organizer ID, $\mathbf{u}_{m-1}$, $\mathbf{v}_{m-1}$, $\boldsymbol{\delta}_m$)

update $\bar{c}_{ij}\ \forall j$ with received $\mathbf{u}_{m-1}, \mathbf{v}_{m-1}$,$\boldsymbol{\delta}_m$
if $\{j \mid \bar{c}_{ij} = 0\} \neq \varnothing$ **then**
  for each j of $\{j \mid \bar{c}_{ij} = 0, j \neq a(i)\}$ **do**
    send $(\mathbf{u}_{m-1}, \mathbf{v}_{m-1}, \boldsymbol{\delta}_m)$ to ID $a^{-1}(j)$
  send newly involved IDs and No. of new branches to organizer
else
  send $\min\{\bar{c}_{ij} | \bar{c}_{ij} > 0, \forall j\}$ to organizer, request dual adjustment

Once a reduced cost becomes feasible it never becomes infeasible again (see Theorem 5.2) so the algorithm needs to iteratively transform each infeasible reduced cost to approach global optimality. Two different approaches for locating and transforming the infeasible values lead to two versions of the algorithm: *task-oriented* and *robot-oriented* variants.

A. Task Oriented Variant

The task oriented approach attempts to cover all infeasible reduced costs of one task before moving to the costs of other tasks; it is, thus, operates *column-wise* in the cost matrix. The task oriented approach mimics the procedure of the centralized version: for any given task (column), the robot holding the smallest projected infeasible reduced cost is elected as organizer. During the swap loop searching stages, it is possible that after some DUAL_ADJs one members can hold a "worse" projected infeasible reduced cost. Therefore, after each update of $\boldsymbol{\delta}$, the organizer must check all involved members within the current tree, and hands over the organizer role if necessary.



(a) (b)

Fig. 5. Illustrations of task oriented (a) and robot oriented (b) strategies. Here shaded entries have infeasible reduced costs. Solid and void stars represent current starting entry and (possibly) next starting entry, respectively.

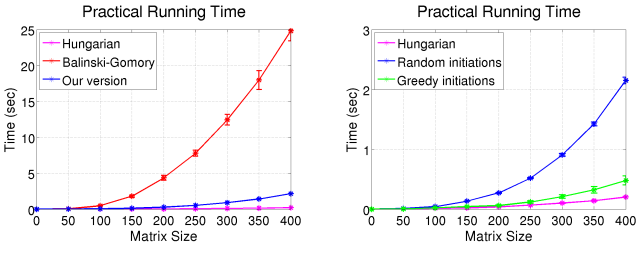
B. Robot Oriented Variant

The robot oriented method aims to covering all infeasible reduced costs of one robot before transferring to another robot; it works in a *row-wise* fashion. The organizer is randomly selected from all members that hold infeasible reduced costs, and keeps the role for the whole stage. Monitoring of "worse" projected costs is not required, but the each stage only guarantees that the starting entry will become feasible, not others. This means the organizer may need to iteratively fix all its associated infeasible reduced costs at each stage before transferring the role to a successor organizer. ◁

To compare, (*A.*) the advantage of the task-oriented scheme lies in that at most n stages are needed to reach global optimality, since each stage turns all infeasible reduced costs to feasible associated with a task. Its disadvantage is the extra communication because at the beginning of each stage, the member holding the smallest reduced cost for the chosen task has to be determined; additional communications are involved in the monitoring aspect too. (*B.*) the robot oriented strategy has greater decentralization and eliminates extra monitoring communication (disadvantage mentioned in the task oriented scheme). At any stage only a subset of robots need be involved and no global communication is required. The disadvantage of this variant is that a total of $O(n^2)$ stages (note, each stage is equivalent to $O(n)$ steps of Balinski-Gomory's method) is needed.

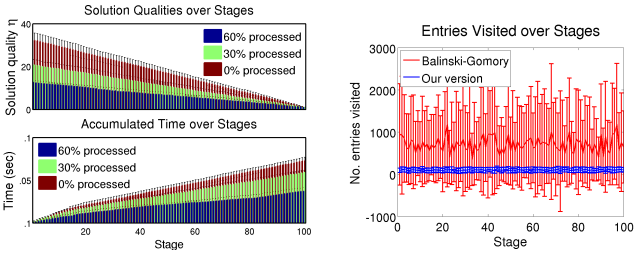
VII. EXPERIMENTS

Three forms of experiment were conducted: run-time performance of the centralized algorithm, access pattern analysis, and comparison of the decentralized variants.



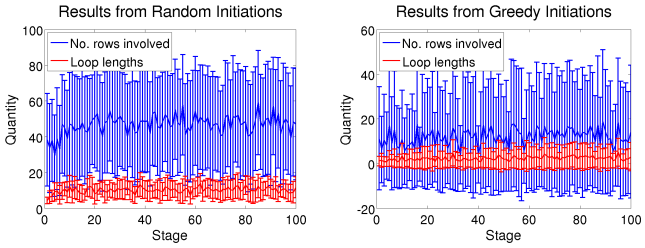
(a) (b)

Fig. 6. Comparison of running times: (a) Time from an optimized Hungarian method, the Balinski-Gomory's method, and the swap-based algorithm. Primal methods start with random initial solutions; (b) Running time is improved when the algorithm is combined with a fast approximation method.



(a) (b)

Fig. 7. (a) Linear solution quality and running time from different initial solutions (matrix size: 100×100); (b) Entries traversed during stages.



(a) (b)

Fig. 8. Quantities of involved rows (robots) and lengths of swap loops over stages (matrix size is 100×100). (a) Results from random initial solutions; (b) Results from greedy initial solutions.

A. Algorithmic Performance Analysis

We implemented both our swap-based algorithm and Balinski-Gomory's method in C++ (with STL data structures), and used an optimized implementation of Hungarian algorithm ($O(n^3)$ complexity) available in the dlib library (`http://dlib.net`) for comparsion. The experiments were run on a standard dual-core desktop with 2.7GHz CPU with 3GB of memory. Fig. 6(a) shows the performance results. We can see that the swap-based algorithm has a significantly improved practical running time over the Balinski-Gomory's method. The flexibility of the algorithm allowed for further improvement: fast approximation algorithms can give a reasonable initial assignment. Fig. 6(b) shows the improvement using an extremely cheap greedy assignment that assigns the robot-task pairs with lowest costs first, in a greedy manner. This reduces the practical running time to be very close to the Hungarian algorithm, especially for matrices with $n < 300$.

To analyze solution quality as a function of running-time, we computed scenarios with 100 robots and 100 tasks with randomly generated $c_{ij} \in [0, 10^4]\ \forall i, j$. The solution qualities and consumed time for individual stages is illustrated in Fig. 7(a). The solution quality is measured by parameter η calculated as a ratio of current solution α_i at current stage i to the final optimum α_n, *i.e.*, $\eta = \alpha_i/\alpha_n \geq 1$. In each figure, the three series represent initial assignments with different "distances" to the optimal solution. A 60% processed initial solution means the initial solution is α_{60} (the solution output at 60$^{\text{th}}$ stage from a random initialization). The matrix is column-wise shuffled before the input of a processed solution such that a new re-computation from scratch can be executed (otherwise it is equivalent to the continuing computation). We can see that the solution qualities for all three scenarios change approximately linearly with the number of stages, which indicates the "step length" for the increment is a constant. From this observation, *computational resources and solution accuracy are fungible* as each is controllable in terms of the other. Given a current solution α_m at the m^{th} stage ($m \geq 1$) as well as an initial solution α_0, the optimum can be estimated:

$$\hat{\alpha}_n = \alpha_0 + \frac{n}{m}(\alpha_m - \alpha_0) = \alpha_0 + n\Delta_s, \tag{15}$$

where $\Delta_s \geq 0$ is the step length of solution increment. To bound the accuracy within $1 + \epsilon$, where $\epsilon \geq 0$, assume we need to stop at θth stage, then

$$\frac{\alpha_0 - \theta\Delta_s}{\hat{\alpha}_n} \leq 1 + \epsilon \implies \theta \geq \frac{\alpha_0 - (1+\epsilon)(\alpha_0 - n\Delta_s)}{\Delta_s}. \tag{16}$$

B. Access Patterns Imply Suitability for Distribution

Intuitively, entries in the spanning tree during each stage reflect the cost of communication.* Thus, we compared the access pattern of our swap-based algorithm with Balinski-Gomory's method on 100×100 matrices with random initial assignment as used. Fig. 7(b) shows that swap-loop traversal results in a large reduction in accesses: the average is ~ 100 for each stage, in contrast with Balinski-Gomory's method requiring $\sim$ 700 with larger standard deviations (actually, reaching more than $8,000$ traversals when many dual adjustments occur). The results quantify claims made about the swap-based method fitting a decentralized paradigm.

We also investigated the total number of rows (and, correspondingly, columns) involved during each stage, which reflects the number of involved robots in decentralized applications, as well as the size of swap loop formed at the end of the stages (defined as the number of colored rows). Fig. 8 show results from random (left plot) and greedily (right plot) initiated solutions. We see that the number of involved rows can be significantly reduced given better initial solutions, and loops are comparatively small for either cases.

More detailed statistics are given in the table below. We conclude that improving initial assignment solutions, not only improves running time, but also the degree of locality in communication and computation. The averaged longest swap lengths show that the admissible primal transformations are a series of small swaps (one can regard the longest length equivalent to K of K-swaps), but which still attains optimality.

STATISTICS OF SWAP LOOPS AMONG STAGES (MATRIX SIZE 100)

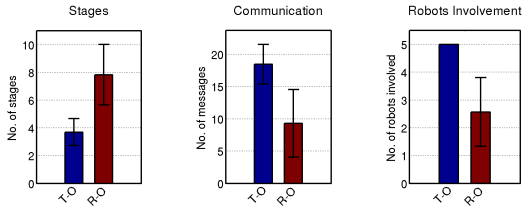
Initial solution	No. loops	Avg. length	Avg. longest	Avg. involved
random initiation	97.12	10.16	21.06	46.72
30% processed	71.90	7.34	19.97	34.29
60% processed	47.20	4.46	14.56	23.92
greedy initiation	24.86	2.30	11.80	16.14

Note: The last three columns denote the averaged lengths, the averaged longest lengths of swap loops, and the averaged number of colored rows in single stages, respectively.

C. Results from Decentralized Variants

We also implemented both variants of the decentralized algorithms and distributed them over five networked computers for testing. The implementations can be directly applied to distributed multi-robot task-assignment, *e.g.*, as the test routing problems in [24]. The hosts were given unique IDs from 1 to 5, and communication performed via UDP, each host

*Every traversed entry on the path segments, no matter it is assigned or unassigned, must connect to a new entry in other rows, requiring a message be passed. The number is approximately half of all the traversed entries since each entry is counted twice for the analysis of communication complexity.



(a) (b) (c)

Fig. 9. Performance of the task oriented (T-O) and robot oriented (R-O) decentralized implementation. Measurements of 5 hosts to 5 tasks.

running a UDP server to listen to the messages sent by its peers. Information such as the IDs of machines, values of dual variables, requests of dual adjustments, *etc.*, were encoded via simple protocols over the message passing. To initiate the system, we inject 5 tasks with IDs from 1 to 5 and each machine randomly generates an array of cost values associated with these 5 tasks. The initial allocation assigns every machine with ID to the task with the identical ID; the corresponding costs for these assigned pairs are communicated. An initial organizer is randomly selected.

Both distributed variants of the algorithm were tested. Fig. 9(a) shows the number stages used for the two schemes (average and variance for 10 separate instances). Fig. 9(b) and Fig. 9(c) show the communication cost (number of messages) and robots involved (ever having received/processed messages) per stage, respectively. These empirical results also validate the claims made above: (*i*) the task oriented scheme requires fewer stages, but has greater communication per stage; (*ii*) although the robot oriented method uses more stages, less the communication and fewer the robots are involved, indicating more local computation and communication.

VIII. CONCLUSION

Strategies of task swaps are a natural paradigm for decentralized optimization and have been used for years (and identified independently by several groups). It is now, using the algorithm we present, that optimality can be guaranteed with these same primitive operations. Additionally, we have sought to emphasize the useful any-time aspect of primal techniques.

In summary, we highlight features of the introduced method:

Natural primitives and optimality: the method is based on task swap loops, a generalization of O-contracts, task-exchanges, and K-swaps; these are techniques which have intuitive interpretations in distributed systems and natural implementations. However, unlike other swap-based methods, global optimality can be guaranteed.

Computational flexibility and modularity: the algorithm can start with any feasible solution and can stop at any non-decreasing feasible solution. It can be used as a portable module to improve non-optimal assignment methods, *e.g.*, some variants of market-based, auction-like methods.

Any-time and efficiency: Unlike primal techniques for general LPs, optimality is reached within strongly polynomial time. Initialization with fast approximation methods makes it competitive practically, and it can potentially be further accelerated. Additionally, the linear increase in the solution quality makes balancing between the computation time and assignment accuracy possible.

Ease of implementation: the algorithm uses simple data structures with a straightforward implementation that is much simpler than comparably efficient techniques.

Ranked solutions: assignments are found with increasing quality, allowing fast transitions to good choices without re-computation if commitment to the optimal assignment fails.

Decentralized Variants, Local Computation & Communication: a small subset of robots are found to be typically involved. The decentralized variants of the algorithm require no single privileged global controller. They allow one to choose to trade between decentralization (communication) and running time (number of stages).

REFERENCES

[1] S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems," *AI Magazine 17(3)*, 1996.

[2] L. E. Parker, "Multiple Mobile Robot Systems," in *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008, ch. 40.

[3] 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*, 1997, pp. 69–82.

[4] M. B. Dias, , and A. Stentz, "Opportunistic optimization for market-based multirobot control," in *Proc. IROS*, 2002, pp. 2714–2720.

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

[6] T. Sandholm, "Contract types for satisficing task allocation: I Theoretical results," in *AAAI Spring Symp: Satisficing Models*, 1998, pp. 68–75.

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

[8] X. Zheng and S. Koenig, "K-swaps: cooperative negotiation for solving task-allocation problems," in *Proc. IJCAI*, 2009, pp. 373–378.

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

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

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

[12] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems: A tutorial," *Interfaces 20(4):133–149*, 1990.

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

[14] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *IJRR 23(9):939–954*, 2004.

[15] M. Nanjanath and M. Gini, "Dynamic task allocation for robots via auctions," in *Proc. ICRA*, 2006, pp. 2781–2786.

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

[17] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-Based Multirobot Coordination: A Survey and Analysis," *Proc. of the IEEE*, 2006.

[18] S. Koenig, P. Keskinocak, and C. A. Tovey, "Progress on Agent Coordination with Cooperative Auctions," in *Proc. AAAI*, 2010.

[19] 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.

[20] G. Dantzig, *Linear Programming and Extensions*. Princeton University Press, Aug. 1963.

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

[22] W. Cunningham and I. A.B. Marsh, "A Primal Algorithm for Optimum Matching," *Mathematical Programming Study*, pp. 50–72, 1978.

[23] M. Akgül, "The linear assignment problem," *Combinatorial Optimization*, pp. 85–122, 1992.

[24] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. J. Kleywegt, "Robot Exploration with Combinatorial Auctions," in *Proc. IROS*, 2003, pp. 1957–1962.