An Anytime Assignment Algorithm: From Local Task Swapping to Global Optimality

Lantao Liu and Dylan A. Shell
Department of Computer Science and Engineering
Texas A&M University
College Station, Texas, USA
Email: {lantao,dshell}@cse.tamu.edu

Abstract

The assignment problem arises in multi-robot task-allocation scenarios. Inspired by existing techniques that employ task exchanges between robots, this paper introduces an algorithm for solving the assignment problem that has several appealing features for online, distributed robotics applications. The method may start with any initial matching and incrementally improve the current solution to reach the global optimum, producing valid assignments at any intermediate point. It is an any-time algorithm with a performance profile that is attractive: quality improves linearly with stages (or time). Additionally, the algorithm is comparatively straightforward to implement and is efficient both theoretically (complexity of $O(n^3 \lg n)$ is better than many widely used solvers) and practically (comparable to the fastest implementation, for up to hundreds of robots/tasks). The algorithm generalizes "swap" primitives used by existing task exchange methods already used in the robotics community but, uniquely, is able to obtain global optimality via communication with only a subset of robots during each stage. We present a centralized version of the algorithm and two decentralized variants that trade between computational and communication complexity. 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. Thus, 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.
Keywords: multi-robot task allocation, decentralized assignment, anytime algorithms, task swapping

1 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 total resulting cost and hence the performance of the team. Although the importance of solutions that trade the quality of results against the cost of computation has been established for some time (*e.g.*, the review in Zilberstein (1996)), the assignment problem underlying efficient task-allocation has received little attention in this regard.

While much work on assignment (or weighted matching) algorithms seeks to reduce overall execution time and time complexity, the multi-robot task allocation setting has several features requiring special consideration. These are the result of an inherently distributed system which must tolerate failures in order to achieve robustness in a dynamic operating environment. For example, a single central irreplacable controller should not be relied upon to compute and broadcast the assignment solutions but, instead, computation should be carried out in a distributed way so that individual failures do not affect the whole system. In addition to handling dynamics, emergencies, and unexpected contingencies smoothly, a computationally flexible coordination algorithm would enable the robot system to maintain a real-time response rates. While classic optimal assignment algorithms go some way toward addressing the task allocation problem for multi-robot systems, new solutions are needed for highly-dynamic distributed situations.

This paper introduces an algorithm that computes the optimal allocation of tasks to robots by employing a task swapping mechanism * : after an evaluation robots may opt to exchange their currently assigned tasks. This swap primitive facilitates easy interpretation of the allocation algorithm; compared with other optimal assignment algorithms (*e.g.*, the matching graph based Hungarian method and its variants), the proposed method seems to be much easier to comprehend in totality, easier to understand during its execution, and simpler to implement. Several attractive properties of this task swapping technique have been revealed. For instance, first, the algorithm yields a feasible allocation at any point in its execution, and the assignment is globally optimal once the algorithm has run to completion. This means that the algorithm can be terminated at *any time* before reaching the optimum and the robots will still have a meaningful assignment to tasks. The algorithm is *flexible* (in terms of computation and accuracy) because one can control the running time or level of optimality by selecting the number of stages. Results presented below 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

*The algorithm was first presented in the 2012 Robotics: Science and Systems conference (Liu and Shell, 2012a).

of computation to be estimated. Additionally, the algorithm may start from any initial assignment solution so it can be easily used to refine sub-optimal assignments computed by other methods.

Unlike several popular competing algorithms that compute optimal assignments, the method we present has a decentralized flavor: any stage of computation naturally involves only a small subset of the robots and tasks, communication with other parts of the system may not even be required. The proposed method does assume, however, that at each stage, the selected subset of robots can communicate with one another in some way, either via direct communication (when robots are within communication range, or when global broadcast communication exists) or a multi-hop network (if only local communication is available). For each stage, the algorithm also needs an *organizer* robot to locate necessary members to form a party for swapping tasks. The algorithm's organization and communication is decentralized; with the terms of Cao *et al.* (1997), each stage fits the *locally centralized* paradigm.

The flexibility afforded by an any-time algorithm will be counterproductive if it comes at too high a cost. Fortunately, 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. An additional benefit is that the cost can be borne by multiple robots because variants of the algorithm may be executed in a decentralized way. We are unaware of another solution to the assignment problem with these features.

To organize the paper, we first formulate the assignment problem in Section 3 and show in Section 4 that the process of solving an optimal assignment problem can be decomposed into a series of local task swaps, and prove that existing opportunistic local task swapping methods can lead to sub-optimality. Then in Section 5 we develop a primal method with multiple task swapping iterations, which we show is a re-interpretation and re-design of Balinski-Gomory's method. Through this presentation, we uncover the decentralized nature intrinsic to this framework for computing assignments. Detailed algorithmic descriptions are also provided to ease direct implementation of the method. In Section 6 two decentralized variants of the algorithm are designed. Trade-offs between the time complexity and solution quality are observed both from theoretical analysis and experiments in Section 7.

2 Related Work

Task allocation is one of the central problems in distributed multi-robot coordination (Parker, 2008). In a multi-robot system, robots need to not only take into account the presence of other team members but also to cooperate with them so as to achieve the best performance of the whole system. Different assignment models have arisen in order to formulate and address differing task allocation scenarios (see Gerkey and Matarić (2004) for a review). An important dimension within the task allocation taxonomy is the cardinality of the mapping between robots and tasks, *viz.*, whether the assignment relationship between

robots and tasks is one-to-one, one-to-many, many-to-one, or many-to-many. In this work, we are interested in the problem of exclusively assigning every robot with a unique task (one-to-one mapping), which is the most fundamental and probably most widely investigated assignment problem.

This paper draws a connection between methods for (*i.*) improving local performance, *e.g.*, via incremental clique preferences improvement, and (*ii.*) allocation methods which seek to solve (or approximate) the global optimum of the assignment.

2.1 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 (Golfarelli *et al.*, 1997; Dias *et al.*, 2002; Thomas *et al.*, 2004), also called O-contracts (Sandholm, 1998), task switching (Sariel and Balch, 2006; Wawerla and Vaughan, 2009), and task exchanges (Chaimowicz *et al.*, 2002; Farinelli *et al.*, 2006). These intuitively appealing methods allow for a form of localized, light-weight coordination of the flavor advocated by Stone *et al.* (2010). Among these existing methods, those using task swaps for task allocation are most relevant to this work. Task (or token) swapping among robots have been mentioned in recent work of Farinelli *et al.* (2006); Zheng and Koenig (2009), *etc.* The main idea of the current task swapping methods is that if the system cost can be reduced through exchanging tasks between a pair of robots (or, in a generalized form, among a group of robots), then the assignment is adjusted so those corresponding tasks are transfered to improve the global solution quality. To date most existing task swap algorithms for task allocation are understood and explained on the basis of intuition alone. One exception is recent theoretical analysis (*e.g.* see Zheng and Koenig (2009)) showing some properties for certain swapping contracts, called K-swaps. (The constant parameter K is important for the solution quality, but it not clear how it is to be determined in practice; Theorem 4.4 throws some light on the significance of this.) Other interesting properties such as the optimality and the running time still lack sound analysis. The method we present gives new insight into how generalized swap-like mechanisms may ensure optimality; in our case this is through something analogous to automatic computation of the necessary value of K. We are also able to characterize the running-time of the new method.

2.2 Optimal Assignment in Task-Allocation

The first and best-known optimal assignment method is Kuhn's $O(n^3)$ Hungarian algorithm (Kuhn, 1955). It is a *dual-based* (or generally *primal-dual*) algorithm because the variables in the dual program are maintained as feasible during each iteration whilst a primal solution is sought. Many other assignment algorithms have been developed subsequently (see the review in Burkard *et al.*

(2009)). Most of them are dual-based methods including: augmenting path (Edmonds and Karp, 1972), the Auction (Bertsekas, 1990), and pseudo-flow (Goldberg and Kennedy, 1995) algorithms, *etc.* These (and approximations to them) underlie many demonstrations of multi-robot task-allocation, *e.g.,* see Gerkey and Matarić (2004); Nanjanath and Gini (2006); Giordani *et al.* (2010). Special mention must be made of market-based methods (*e.g.,* Dias *et al.* (2006); Koenig *et al.* (2010)) 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 (Bertsekas, 1990). Zavlanos *et al.* (2008) extended Bertsekas's auction algorithm and introduced a distributed variant in the context of networked systems. Fully distributing the auction based methods sacrifices optimality: Lagoudakis *et al.* (2005) gives bounds for some auction strategies applied in the distributed multi-robot systems.

Comparatively few papers report use of *primal* approaches for task-allocation; researchers who solve the (relaxed) Linear Program directly likely use the popular (and generally non-polynomial time) simplex method (Dantzig, 1963). The primal assignment algorithm proposed by Balinski and Gomory (1964) is a little known method that appears entirely unheard-of within robotics. The relationship to the present work is not obvious from their presentation, but their chaining sequence of alternating primal variables is analogous to the swap loop transformation we have identified. The centralized algorithm we present 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 concern themselves with.

3 Problem Description and 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 Gerkey and Matarić (2004). An assignment $\mathcal{A} = \langle R, T \rangle$ consists of a set of robots R and a set of tasks T. Let matrix $C = (c_{ij})_{n \times n}$, where $c_{ij} : R \times T \to \mathbb{R}^+$ represents the cost of having robot i perform task j. Without loss of generality, in our work, $n = |R| = |T|$, the number of robots is identical to the number of tasks (otherwise dummy rows/columns can be inserted).

3.1 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 (so each 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, so 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 $\mathbf{v} = \{v_j\}$, with dual linear program:

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

3.2 Reduced Cost, Complementary Slackness, and Feasibility

*Reduced cost*s are defined as follows

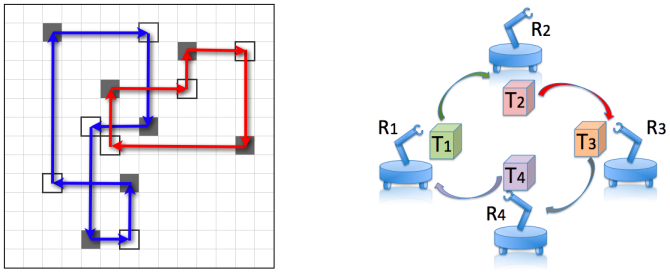
$$\bar{c}_{ij} = c_{ij} - u_i - v_j, \qquad \forall (i,j). \tag{3}$$

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

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

$$x_{ij}(c_{ij} - u_i - v_j) = x_{ij}\bar{c}_{ij} = 0, \qquad \forall (i,j). \tag{4}$$

Equation (4) is called *complementary slackness* condition, which 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.



(a) (b)

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

3.3 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 $|\{(i,j) \mid \bar{c}'_{ij} \geq 0\}| > |\{(i,j) \mid \bar{c}_{ij} \geq 0\}|$.

4 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 arrows, each of which bridges exactly one shaded entry (old assignment) and exactly one bold-edged entry (new assignment). Note that both types of such entries have reduced costs equal to 0s. Connecting the beginning of the chain to its end closes the path, forming what we call a *swap loop*. They are 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.

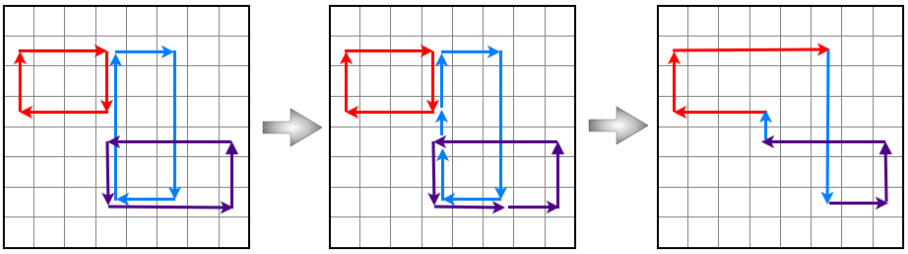


Fig. 2: An amalgamation allows synthesis of complex swap loops from multiple dependent swap loops. Overlapped path segments cancel each other out.

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 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, then the i^{th} column of $\mathbf{P}$, $\mathbf{p}_i = \mathbf{e}_i$ (the i^{th} column of the identity matrix) and $\mathbf{P} = \prod_{\lambda=1}^{m} \mathbf{P}_\lambda$, 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 the product does not matter. Thus we have $\mathbf{X'} = \mathbf{PXQ} = \mathbf{P_1 P_2 \cdots P_m X Q_m Q_{m-1} \cdots Q_1}$ (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 cannot 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 vice versa.*

Proof. A directed path segment can be conveniently represented as vector $\vec{\pi}$, as shown in Fig. 2. Overlapping path segments $\vec{\pi_1}$ and $\vec{\pi_2}$ in 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 dependent 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 objective of the whole assignment problem can be written in the form

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

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

$$\begin{aligned} \binom{n-1}{K-1}^{-1} \sum_{\gamma \in [1, |S_{\mathcal{A}}|]} \max\ g(\mathcal{A}_\gamma) &\geq \max \sum_{\gamma \in [1, |S_{\mathcal{A}}|]} \binom{n-1}{K-1}^{-1} g(\mathcal{A}_\gamma) \\ &= \max\ g(\mathcal{A}) \end{aligned} \tag{8}$$

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

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

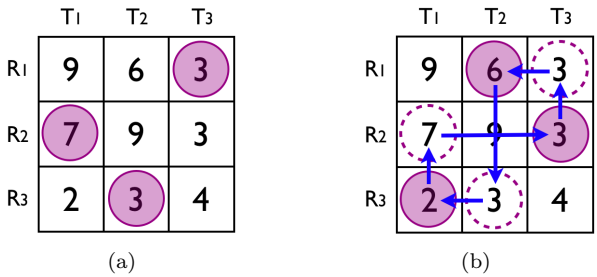


Fig. 3: Illustration of the sub-optimality of K-swaps. (a) A cost matrix with sub-optimal assignment solution (shaded in circles) after sufficient 2-swaps; (b) The optimal solution can only be obtained via a 3-swap, indicating K must be 3 ($= n$) in this case.

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

□

The assignment in Fig. 3 shows an example of the sub-optimality. The cost matrix is constructed from 3 robots (row indices) and 3 tasks (column indices). In Fig. 3(a) the current assignment solution is denoted by those shaded entries. Assume that the robots can only do a series of 2-swaps ($K = 2$), then there are $\binom{3}{2} = 3$ possible pair-wise swaps among the robots. Fig. 3(a) shows a local minimum since no further 2-swaps can proceed. The optimal solution is shown as the shaded entries in Fig. 3(b). To transfer from the assignment in Fig. 3(a) to the optimal solution, a 3-swap has to be executed, illustrated as the loop in Fig. 3(b).

5 An Optimal Swap-based Primal Method

The complementary slackness in (4) indicates that if a robot-task pair (i, j) is assigned, then the corresponding reduced cost $\bar{c}_{ij}$ must be equal to 0. Moreover, no $\bar{c}_{ij}$ can have a negative value due to the constraint in Program (2). The essence of the proposed task swapping method is that, as the algorithm proceeds, those infeasible (negative) $\bar{c}_{ij}$ are successively turned into feasible ones. While this proceeds, there is always a set of mutually excluded entries with reduced costs equal to zero, so that an assignment solution is always known, although it might be different from stage to stage.

In fact searching for a swap loop is actually precisely the procedure to seek a better feasible assignment solution so as to supersede the current assignment. To be precise "better" here is in the sense of improved (admissible) primal and dual transformations.

The results in Section 4 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 this iteratively and avoids local minima because later swaps may correct earlier ones based on "enlarged" views that take into account all rows and columns that have been examined so far. 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 is obtained. Once all reduced costs are feasible, the primal and dual solutions simultaneously reach their (identically valued) optimum.

Algorithm 5.1 PRIMAL ($\mathbf{C}$)

```
1: init arrays u[n] := {0}, v[n] := diag(C), C̄[n][n] := {0}
2: for  i := 1 to n do
3:    update matrix C̄ with c̄_{i'j'} = c_{i'j'} − u_{i'} − v_{j'}, ∀i', j'
4:    if min{C̄[:][i]} < 0 then
5:       array δ[n] := {0}
6:       heap h[n] := PRE-PROCESS(C̄, u, v)
7:       check the ith column of C̄, get smallest-valued entry (x, y)
8:       SWAP_LOOP(C̄, δ, h, x, y)
9:       for  j := 1 to n do
10:         u[j] := u[j] + δ(a(j)), v[j] := v[j] − δ[j]
11:      v[y] := v[y] − |C[x][y] − u_x − v_y| so that C̄[x][y] = 0
12:      if a swap loop found, swap tasks to augment solution
```

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

The algorithm is organized as follows: the outline of the procedure is in Algorithm 5.1, and critical steps are described in Algorithms 5.2–5.4 in some detail to ensure that the pseudo-code is appropriate for straightforward implementation. In Algorithm 5.1, there are at most n *stage*s, each of which searches for a swap loop to obtain an admissible primal transformation (note that this may trigger the dual-update operation during the re-searching step). Since every primal transformation is admissible, the assignment solution never deteriorates and will actually improve with each iteration.

5.1 Algorithm V.2: 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, so that smallest values (root elements) can be *extracted* or *removed* efficiently.

Algorithm 5.2 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
```

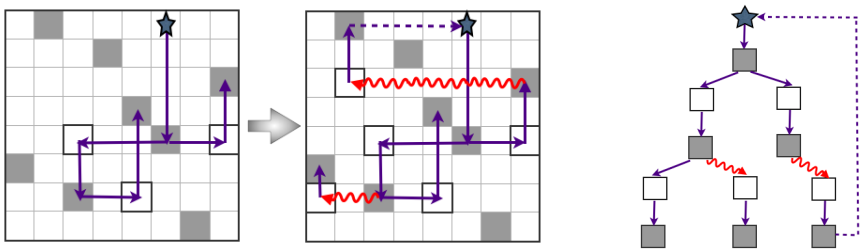
5.2 Algorithm V.3: Searching for Swap Loops

Any swap loop yields an admissible primal transformation. Loops are sought by bridging path segments in the reduced cost 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. If there are multiple zero-valued reduced costs in the same row, multiple horizontal path segments are built by connecting their corresponding entries to the unique assigned entry in that row. Vertical path segments are implicitly identified from the entries of the zero-valued reduced costs to the uniquely assigned entries in the respective columns. Fig. 4(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 5.3 SWAP_LOOP ($\bar{\mathbf{C}}$, $\mathbf{h}$, $\boldsymbol{\delta}$, x, y)

```
 1: starting row r_s := x, column t_s := y, S_R := S_T := ∅
 2: initiate r := a^{-1}(y), t := y, Vpath(t : r_s → r)
 3: push r into queue Q, color(S_R ∪ {r}; S_T ∪ {t_s})
 4: while Q not empty AND Q.front ≠ r_s do
 5:    r := Q.front, Q.pop once
 6:    initiate set S_δ := {r}
 7:    for each r ∈ S_δ  do
 8:       t := h[r].extract.label
 9:       while p(r, t)† = 0 do
10:          if t ∉ S_T then
11:             Hpath(r : a(r) → t), Vpath(t : r → a^{-1}(t))
12:             push a^{-1}(t) into Q, color(S_R ∪ {a^{-1}(t)}; S_T ∪ {t})
13:          h[r].remove root element and update root
14:          update t := h[r].extract.label
15:    if Q empty then
16:       DUAL_ADJ (C̄, Q, h, δ, S_δ, r_s, t_s)
17:       if updated S_δ not empty then
18:          go to STEP 7
19:       return
20: Hpath(r_s : t → t_s), form a loop
```

†: $p(\cdot)$ is a projection of reduced cost, defined in (12) on page 14.



(a) (b)

Fig. 4: (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.

In Algorithm 5.3, function $t = a(r)$ denotes the assignment 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 $Hpath(cur\_row : col1 \rightarrow col2)$ ($Vpath(cur\_col : row1 \rightarrow row2)$), 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.

5.3 Algorithm V.4: Dual Adjustments

If all branches reach dead-ends, dual adjustment introduces entries with zero-valued reduced costs so that the tree may be expanded further. This is done by changing the values of dual variables, which indirectly changes the reduced costs of corresponding entries. Doing so must increase the size of the set of feasible reduced costs, so the dual adjustment will never cause the current result to deteriorate. The method subtracts the smallest feasible reduced cost from all visited (colored) rows and adds it to every visited column, producing at least one new zero-valued reduce cost(s). Waved arrows in Fig. 4 show this procedure.

Next, we turn to the relationship of this approach to Balinski-Gomory's primal technique (Balinski and Gomory, 1964). Theoretical complexity and empirical results below show the superiority of the swap-based approach. Nevertheless, examining the conceptual differences in some detail is worthwhile since a common underlying idea is involved: Balinski-Gomory's method employs iterative labeling and updating techniques to seek a chaining sequence of alternating primal variables, which are used to adjust and augment the primal solutions.

By way of contrast, we highlight three aspects of the presented algorithm worthy of highlighting:

1. The swap loop search incorporates the dual adjustment procedure. In each

Algorithm 5.4 DUAL_ADJ ($\bar{\mathbf{C}}$, Q, $\mathbf{h}$, $\boldsymbol{\delta}$, S_δ, r_s, t_s)

```
array top_d[n] := {∞}, col_d[n] := {0}
for  i := 1 to n do
    if row i ∈ S_R then
        top_d[i] := p(i, h[i].extract.label)
min_δ := min{top_d[i]}
if min_δ > 0 then
    update S_δ := {i | top_d[i] = min_δ}
else
    min_δ := −p(r_s, t_s)
for  i := 1 to n do
    if row i ∈ S_R then
        update δ[a(i)] := δ[a(i)] + min_δ
    col_d[i] := p(i, t_s)
if min{col_d[i]} ≥ 0 then
    terminate current stage
update starting row r_s := argmin_i{col_d[i]}
if r_s ∈ S_R then
    Hpath(r_s : a(r_s) → t_s), form a loop
    terminate current swap loop searching
```

stage Balinski-Gomory's method may require n rounds of traversals and cost $O(n)$ more steps. We reduce the traversals by building and maintaining a search tree. Traversals between nodes of different hierarchical levels (depths) become especially convenient and cheap. (New nodes introduced by dual adjustments could be expanded at any level, not necessary at the leaf nodes of the lowest level.) This modification is most significant in the decentralized context as each traversal incurs 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 $\mathbf{v}$ once the whole stage is completed:

$$u'_i = \begin{cases} u_i + \sum_\omega \delta^{(\omega)}_{a(i)}, & \forall i \in S_R \\ u_i \text{ otherwise}, \end{cases} \tag{10}$$

$$v'_j = \begin{cases} v_j - \sum_\omega \delta^{(\omega)}_{j}, & \forall j \in S_T \\ v_j \text{ otherwise}, \end{cases} \tag{11}$$

where S_R and S_T are index sets of colored rows and columns, respectively, and ω is the iteration index. The benefit here lies in that reduced costs in the whole matrix need not be immediately updated on each dual variable's adjustment. Instead, a query of reduced cost $\bar{c}'_{ij}$ for the 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 (this occurs in Step 16 of Algorithm 5.4).

5.4 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 $x_{ij} = 1$ and $c_{ij} = u_i + v_j$ (see (4)). With (10) and (11):

$$\begin{aligned} f(\mathbf{X}') - f(\mathbf{X}) &= \sum_{i,j}(u'_i + v'_j) - \sum_{i,j}(u_i + v_j) \\ &= \sum_i \left(u'_i - u_i + v'_{a(i)} - v_{a(i)}\right) \\ &= \sum_i \left(\sum_\omega \delta^{(\omega)}_{a(i)} - \sum_\omega \delta^{(\omega)}_{a(i)}\right) - |\bar{c}_{kl}| \\ &= \bar{c}_{kl} < 0. \end{aligned} \tag{13}$$

(See Step 11 in Algorithm 5.1.) Thus, the value of the primal objective must decrease after a swap.

□

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:

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

Term $\bar{c}_{ij} - \sum_{\omega} \delta_{a(i)}^{(\omega)} \geq 0$ is because $\sum_{\omega} \delta_{a(i)}^{(\omega)}$ accounts for the smallest feasible reduced costs from all colored rows, so $\sum_{\omega} \delta_{a(i)}^{(\omega)}$ is always less than or equal to $\bar{c}_{ij}$. Moreover, those entries that are neither in S_R nor S_T remain unchanged.

Second, at least $\bar{c}_{kl}$ will become newly feasible, leading to termination before formation of a swap loop. This fact is true even in the more sophisticated strategy allowing dynamic updating of starting entry (see Step 16 of Algorithm 5.4). This proves that the set of feasible reduced costs must increase.

□

5.5 Time Complexity

Algorithm 5.1 requires at most n stages. In each stage the smallest infeasible reduced cost in each column is selected (Step 7 of Algorithm 5.1), so all other infeasible entries in the same column must also become feasible. 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 using the heaps. Visited columns are colored in a sorted set and are never considered for bridging future paths within a stage. There are at most n^2 entries to color and check, each costs $O(\lg n)$, yielding a total of $O(n^2 \lg n)$ per stage. Therefore, the total time complexity for the whole algorithm is $O(n^3 \lg n)$. The light-weight operations involved result in a small constant factor.

By way of comparison, Balinski-Gomory's primal method uses $O(n^2)$ searching steps with $O(n^2)$ time complexity for each step. Some researchers (Cunningham and A.B. Marsh, 1978; Akgül, 1992) have suggested that it may possible to further improve the time complexity to $O(n^3)$ using techniques such as the blossom method (Edmonds and Karp, 1972). To the best of our knowledge, no such variant has been forthcoming.

In addition, we note that although the min-heaps in Algorithm 5.2 are created in a separate step, this is to simplify the description. In practice they can be constructed on the fly when required; although the time complexity is unchanged, one may obtain a better practical running time in this way. Our 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.

6 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 as searching progresses. The idea is illustrated in Fig. 5 with four robots executing a single stage of the swap loop search. The lines in Fig. 5(a) show the initial pairwise robot-task assignment; the arrows in Fig. 5(b) show bridging edges found by searching for a swap loop starting from a selected pair. If the path's ending pair connects to

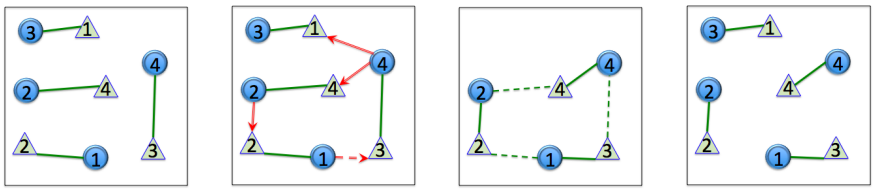


Fig. 5: Swap loop searching in a multi-robot system using the Euclidean distance as a cost metric. Circles represent robots and triangles denote the tasks.

the starting pair, then a swap loop has been found (Fig. 5(c)) and tasks can be exchanged among the robots in the loop. The resulting assignment is shown in Fig. 5(d). Note that the edges connecting to the tasks (triangles) are for illustration purposes, and should not be interpreted as a communication topology since obviously a task cannot pass a message. In reality, the messages are passed and processed by the associated robots who are currently assigned to these tasks. An additional reason for using this graph depiction is that it shows the searching structure shown in Fig. 4 directly, but in the physical space. This graph representation is also regarded as *hypergraph*. It describes the association relationship between robots and tasks, but not the connections among items of the same type. Advantages of the hypergraph view are discussed in Liu and Shell (2012b).

One challenge is that, unlike centralized algorithms, the cost matrix may be not globally visible. Instead each robot maintains and manipulates its own cost vector associated with all the tasks. One 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 decentralized version of the algorithm can be understood in terms of two roles: a *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 in the centralized case. 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 6.1 and 6.2.

Algorithm 6.1 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 6.2 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

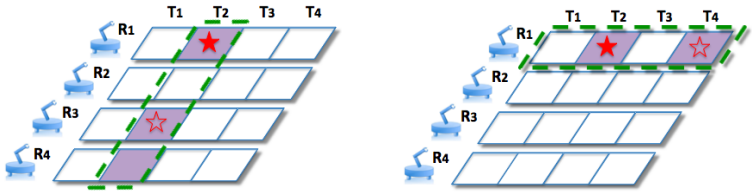
Once a reduced cost becomes feasible it never becomes infeasible again (see Theorem 5.2) so the algorithm needs to iteratively transform each infeasible reduced cost to approach global optimality. Two different approaches for locating and transforming the infeasible values lead to two versions of the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

6.1 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 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 the organizer. During the swap loop searching stages, it is possible that after some DUAL_ADJs other members hold even "worse" projected infeasible reduced costs. Therefore, after each update of $\boldsymbol{\delta}$, the organizer must check all members involved within the current tree, and hand over the organizer role if necessary.

6.2 Robot Oriented Variant

The robot oriented method aims to cover 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; it keeps the role for the whole stage. Monitoring "worse" projected costs is



(a) (b)

Fig. 6: 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.

not required, but each stage only guarantees that the starting entry will become feasible, not the others. This means the organizer needs to iteratively "fix" all its infeasible reduced costs stage by stage before transferring the organizer role to a successor.

Comparing the two variants:

(*A.*) The advantage of the task-oriented scheme is that at most n stages are needed to reach the global optimum, since each stage makes all infeasible reduced costs associated with a task feasible. Its disadvantage is the extra communication involved because at the beginning of each stage, the member holding the smallest reduced cost for the chosen task has to be determined/elected; additional communications are required for the monitoring aspect too.

(*B.*) The robot oriented strategy has greater decentralization and eliminates extra communication for monitoring (a disadvantage mentioned in the task oriented scheme). At any stage only a subset of robots need be involved instead of requiring participation of all robots. The disadvantage of this variant is that a total of $O(n^2)$ stages is needed. (Note, that here each stage is still equivalent to $O(n)$ steps/stages of Balinski-Gomory's method)

7 Experiments

Four forms of experiment were conducted: run-time performance of the centralized algorithm, access pattern analysis, demonstration with a dispatching scenario, and comparison of the decentralized variants.

7.1 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 the Hungarian algorithm ($O(n^3)$ complexity) available in the dlib

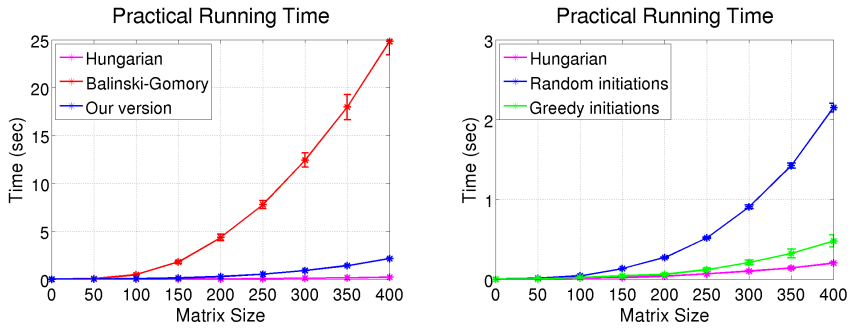


Fig. 7: Comparison of running times: (a) Time from an optimized Hungarian method, the Balinski-Gomory's method, and the swap-based algorithm. Primal methods start with random initial solutions; (b) Running time is improved when the algorithm is combined with a fast approximation method.

library (`http://dlib.net`) for comparison. The experiments were run on a standard dual-core desktop with 2.7GHz CPU and 3GB of memory. Fig. 7(a) shows the performance results. We observe 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. 7(b) shows the improvement using an extremely cheap approximate 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 time consumed for individual stages appear in Fig. 8. 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^{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 must be executed (otherwise it is equivalent to the continuing computation). We see that the solution qualities for all three scenarios change close to linearly with the number of stages, indicating 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

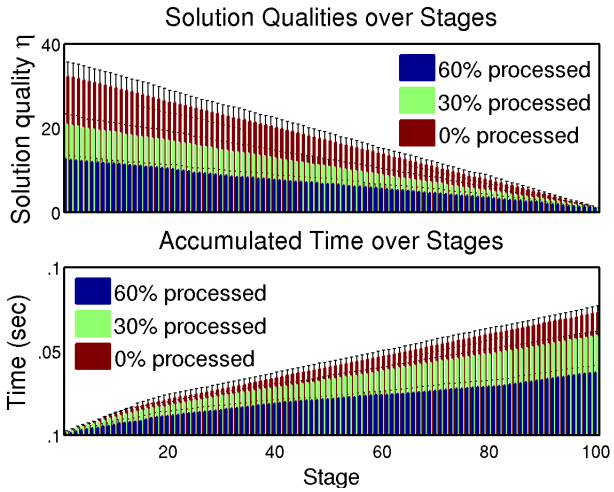


Fig. 8: Linear solution quality and running time from different initial solutions (matrix size: 100×100).

be estimated as

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

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. With following relation

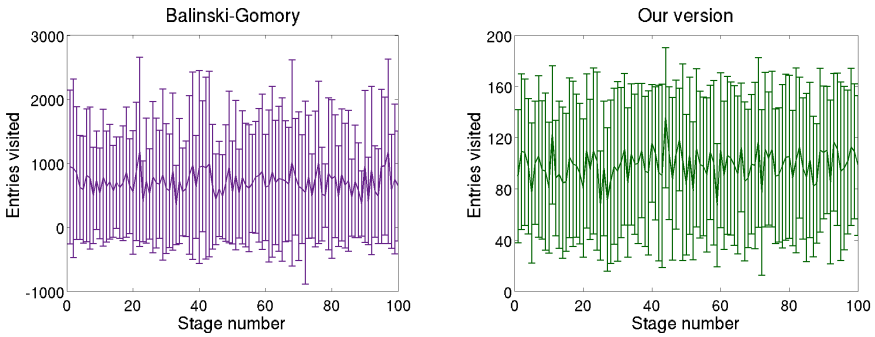
$$\frac{\alpha_0 - \theta\Delta_s}{\hat{\alpha}_n} \leq 1 + \epsilon, \tag{16}$$

finally we get

$$\theta \geq \frac{\alpha_0 - (1+\epsilon)(\alpha_0 - n\Delta_s)}{\Delta_s}. \tag{17}$$

7.2 Access Patterns Show 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. Fig. 9 shows that the swap-loop traversal results in a large reduction in accesses: in our algorithm the average is ~ 100 for each stage, in contrast with Balinski-Gomory's method requiring ~ 700 with larger standard deviations (actually, reaching more than $8,000$ traversals when many dual



(a) (b)

Fig. 9: (a) (b) Entries traversed during stages.

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 loops formed at the end of the stages (defined as the number of colored rows). Fig. 10 shows results from randomly (Fig. 10(a) and 10(b)) and greedily (Fig. 10(c) and 10(d)) initiated solutions. We see that the number of rows involved can be significantly reduced with better initial solutions, and loops are comparatively small for either case.

More detailed statistics are given in the table below. We conclude that improving initial assignment solutions not only improves running time but also reduces the communication costs. 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.

7.3 A Dispatching Scenario

We validated our algorithm in a realistic task allocation problem: dispatching a group of robots to a set of tasks both of which are randomly distributed in space, as shown in Fig. 11. The circles represent robots and the smaller square dots denote tasks (robot and task IDs are labeled on top of them). The costs in the cost matrix are the Euclidean distances between the robots and tasks (in other complex scenarios, *e.g.*, with obstacles in the environment, the costs are the lengths of the planned paths). The left three figures, *i.e.*, Fig. 11(a), 11(c) and 11(e) are the results from random initialization: we initially assign the

[†]Every traversed entry on the path segments, no matter whether 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.

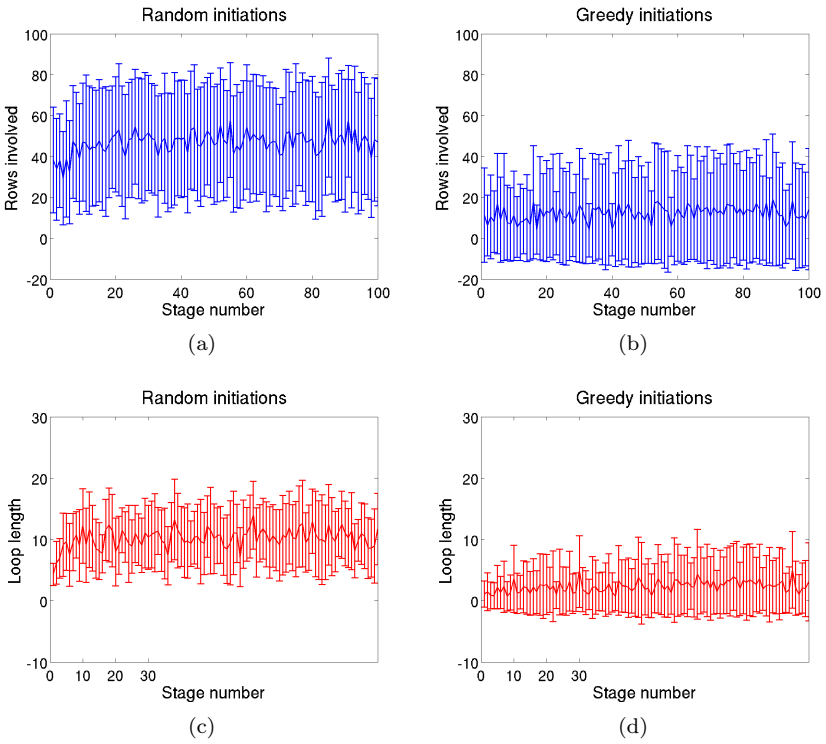


Fig. 10: Quantities of involved rows (robots) and lengths of swap loops over stages (matrix size is 100×100). (a) and (c) Results from random initial solutions; (b) and (d) Results from greedy initial solutions.

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 are averages of lengths, the averaged longest lengths of swap loops, and the averaged number of colored rows in single stages, respectively.

robots to the tasks with the same IDs. In contrast, the right three figures (Fig. 11(b), 11(d) and 11(f)) are results using the greedy initialization: a robot takes the nearest available task, labels it as occupied, until every robot obtains a unique task.

Fig. 11(a) and 11(b) show the final graph constructed during the searching of a swap loop in a particular iteration (the organizers are the 40^{th} robots for both cases). The graph can be regarded as hypergraph as mentioned in Section 6, which displays the searching structure directly in the physical space. The swap loops are highlighted with thickened lines. The two figures reveal that in order to find a swap loop, only a subset of robots (and tasks) need to be involved. Fig. 11(c) and 11(d) are the corresponding cost matrices. The current assignment solutions are denoted by shaded entries in both matrices. We can also see that the shaded entries are generally aligned along the diagonal in Fig 11(c), which reflects the degree to which is close to the "initial" assignment. Path segments are bridged with thin lines and the swap loops are also highlighted in thicker lines. Finally Fig. 11(e) and 11(f) show the communication graph in which communication links were established between robots. The swap loops are also denoted in thick lines. The dashed ellipses outline the range of robots involved in communication. Comparing the figures between left column and right column we find that a good initial solution has the following advantages: it requires lower communication costs as the number of communication links/edges is smaller; the swap loop is also shorter which reduces the costs of task swapping operations; the communication tends to be more localized since the search scope is narrower.

7.4 Results from Decentralized Variants

We also implemented both variants of the decentralized algorithms described in Section 6 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 Berhault *et al.* (2003); Liu and Shell (2011). The hosts were given unique IDs from 1 to 5, and communication performed via UDP, each host 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 injected 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. 12(a) shows the number stages used for the two schemes (average and variance for 10 separate instances). Fig. 12(b) and Fig. 12(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 communica-

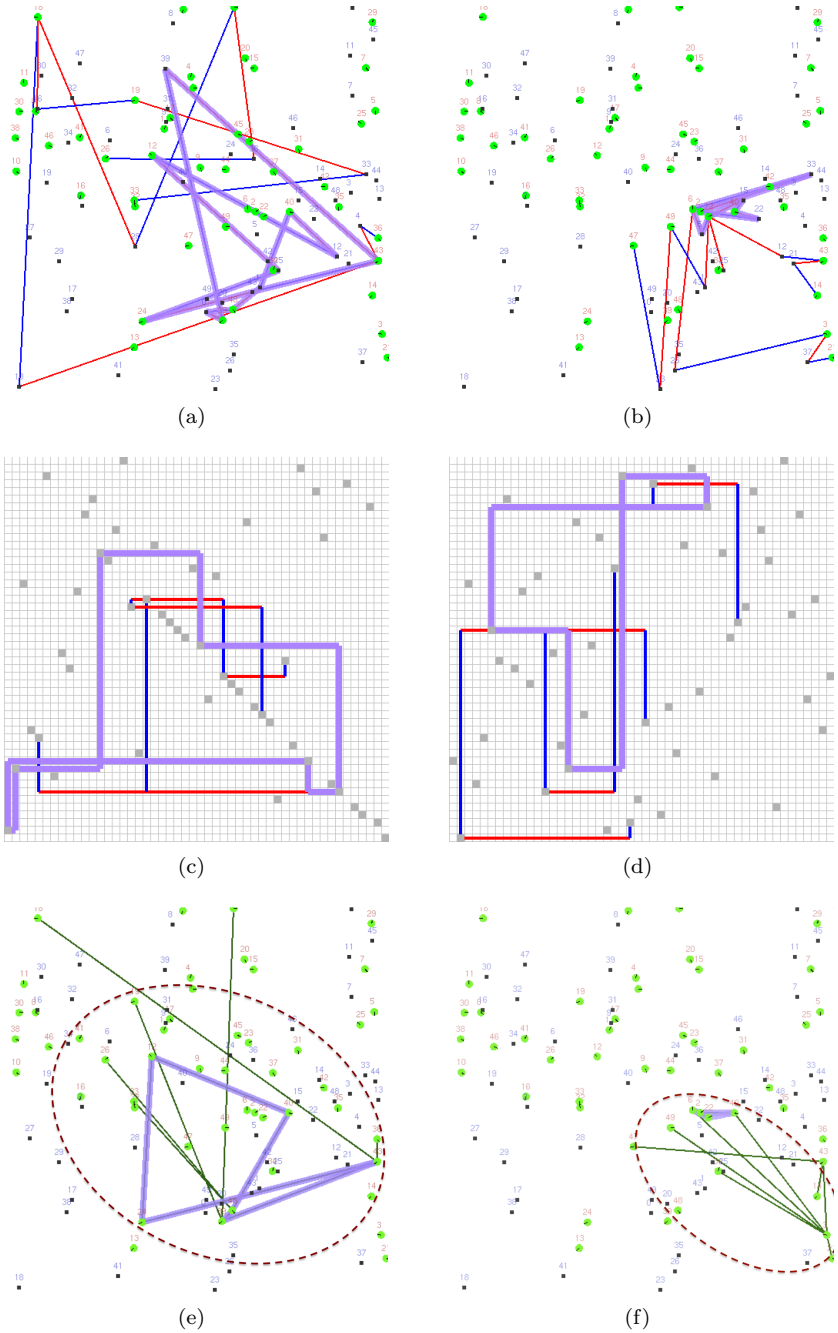


Fig. 11: Task allocation (dispatching) with 50 robots and 50 tasks. (a)(b) Searching a swap loop in the Hypergraph representation; (c)(d) Swap loop searching in the cost matrix; (e)(f) Real communication graph among robots.

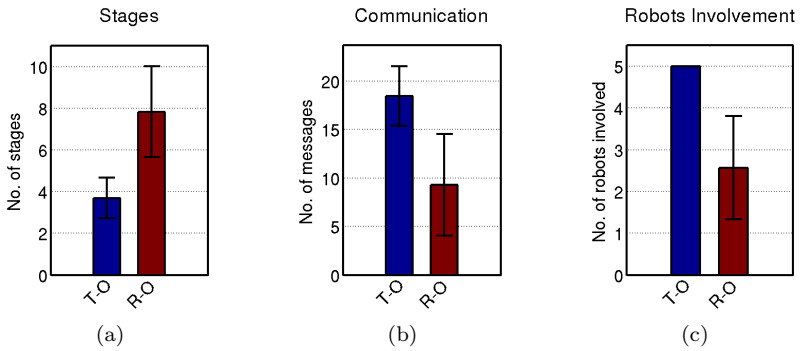


Fig. 12: Performance of the task oriented (T-O) and robot oriented (R-O) decentralized implementation. Measurements of 5 hosts to 5 tasks.

tion per stage; (*ii*) although the robot oriented method uses more stages, less the communication and fewer the robots are involved.

8 Discussion and Future Work

Along with the presentation and analysis of this method, several features have been revealed distinguishing it for other popular assignment algorithms applicable in the distributed multi-robot domains. 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 reached.

- *Computational flexibility and modularity*: the algorithm can start with any feasible solution and can stop at any point. The solution remains feasible and quality is non-decreasing. 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 typically found to be 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).

However, there are still some issues that need to be investigated further, which are also our ongoing and future work. One critical problem is the communication constraints. More specifically, although only a subset of robots can be involved and communicate at each algorithmic stage, this method still requires a complete communication network in which every pair of robots is connected. This may be a strong assumption for many distributed multi-robot systems. For example, some systems can guarantee only local communications within certain ranges and a robot might not be able to establish a communication channel with a distant robot. In such a case searching for a swap loop may fail when the tree cannot be completed to find the loop.

There are a couple of possible ways to improve this algorithm under the communication constraints, which also depend on whether the solution optimality may be sacrificed or not.

- If solution optimality is mandatory then at each stage the tree must be fully constructed so that all reduced costs associated with a particular task are guaranteed to be feasible. Therefore, if a direct communication channel between two robots is not available, multiple hops will be necessary to pass the messages from one end to the other, with multiple robots in middle doing routing work. The communication costs of such methods may increase significantly and an algorithm for efficiently finding the multi-hop routing paths given the form of the tree search has to be designed.

- Is solution sub-optimality is acceptable then the searching tree need not be complete, thus the branches that are not directly reachable can be pruned. Inevitably, the branch pruning will lose the view of the whole problem which may, consequently, sacrifice global optimality.

9 Conclusion

Claims have made in various places about auctions and, by general extension, dual methods being especially suited to multi-robot systems because they are inherently distributable, because the whole cost matrix need not be disclosed to peer agents, and since their treatment of the optimization steps (*i.e.,* modification of dual variables) can be seen as transmitting bids and/or sharing price

information. This paper shows that primal methods can have an equally compelling interpretation: that of task swaps. These swaps are a natural paradigm for decentralized optimization, have been used for years, and identified independently by several groups. It is now, using the algorithm we present, that optimality can be reached with these same primitive operations. Additionally, we have sought to emphasize the useful anytime aspect of primal techniques.

References

Akgül, M. (1992). The linear assignment problem. *Combinatorial Optimization*, pages 85–122.

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

Berhault, M., Huang, H., Keskinocak, P., Koenig, S., Elmaghraby, W., Griffin, P., and Kleywegt, A. J. (2003). Robot Exploration with Combinatorial Auctions. In *Proc. IROS*, pages 1957–1962.

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

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

Cao, Y. U., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, **4**, 226–234.

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

Cunningham, W. and A.B. Marsh, I. (1978). A Primal Algorithm for Optimum Matching. *Mathematical Programming Study*, pages 50–72.

Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton University Press.

Dias, M. B., , and Stentz, A. (2002). Opportunistic optimization for market-based multirobot control. In *Proc. IROS*, pages 2714–2720.

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

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

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

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

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

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

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

Koenig, S., Keskinocak, P., and Tovey, C. A. (2010). Progress on Agent Coordination with Cooperative Auctions. In *Proc. AAAI*.

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

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

Liu, L. and Shell, D. (2011). Assessing Optimal Assignment under Uncertainty: An Interval-based Algorithm. *IJRR 30(7):936–953*.

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

Liu, L. and Shell, D. (2012b). Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots 33(3):291–307*.

Nanjanath, M. and Gini, M. (2006). Dynamic task allocation for robots via auctions. In *Proc. ICRA*, pages 2781–2786.

Parker, L. E. (2008). Multiple Mobile Robot Systems. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, chapter 40. Springer.

Sandholm, T. (1998). Contract types for satisficing task allocation: I Theoretical results. In *AAAI Spring Symp: Satisficing Models*, pages 68–75.

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

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

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

Wawerla, J. and Vaughan, R. T. (2009). 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.

Zavlanos, M. M., Spesivtsev, L., and Pappas, G. J. (2008). A Distributed Auction Algorithm for the Assignment Problem. In *Proc. CDC*.

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

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