Assignment Algorithms for Modeling Resource Contention and Interference in Multi-Robot Task-Allocation

Changjoo Nam and Dylan A. Shell

***Abstract*— We consider optimization of the multi-robot task-allocation (MRTA) problem when the overall performance of the team need not be a standard sum-of-utilities (or costs) model. We introduce a generalization which allows for the additional cost incurred by resource contention to be treated in a straightforward manner. In this variant, robots may have multiple ways to perform a task (*e.g.*, several routes to reach a destination, or use of other shared resources), and interference may be modeled as occurring when multiple robots use the same resource. We show that the general problem is an NP-hard optimization problem. We also investigate instances where the interference results in linear or convex penalization functions.**

We propose an exact algorithm for the general problem. Then we suggest an optimal polynomial-time algorithm and an approximation polynomial-time algorithm for the other problems. The exact algorithm finds an optimal assignment in a reasonable time on small instances. The other two algorithms find an optimal assignment in a short time even if a problem is of considerable size (*e.g.*, in the linear case, 0.5786 sec for 100 robots) and a high-quality solution quickly (*e.g.*, in the convex case, 0.8462 sec), respectively. In contrast to conventional approximation methods, our algorithm provides the performance guarantee.

I. Introduction

The literature on multi-robot coordination has grown enormously in the past few decades, reflecting the potential advantages of a team of robots over a single robot. Multi-robot task-allocation (MRTA) addresses optimization of collective performance by reasoning about which robots in the team should perform which tasks. Even the classical work advocated several different approaches, such as behavior-based [1], [2] and market-based [3], [4] task-allocation. Although resource contention and physical interference have long been known to limit performance [6], the vast majority of MRTA work considers settings for which interference is treated as negligible (*cf.* review in [7]). This limits the applicability of these methods and computing a task assignment under assumptions of noninterference may produce suboptimal behavior even if the algorithm solves the assignment problem optimally. Several authors have proposed task allocation approaches that model or avoid interference (usually physical interference), see for example, [8], [9], [10]. These works, however, do not set out to achieve global optimality, or understand the computational consequences of a model of interference.

In this paper, we assume a centralized system in which all information is known by a dispatcher, which optimizes task allocation. In practice this can be a dynamically elected robot from amongst the team. We also assume that a robot can perform only one task at a time, each task requires only one robot to execute it, and that the allocation of tasks to robots need consider only current (instantaneously available) information and need not hedge against future plans. This problem can be posed as an Optimal Assignment Problem (OAP), which is well-studied, and can be cast as an integral linear program and is in complexity class P. This conventional MRTA problem does not specify how robots use resources so it is unable for it to account for interference incurred by sharing resources. Instead, it assumes that resources are individually allocated to robots or, if shared, that they impose no limits.

In our problem, however, robots may have choice in using resources to perform tasks (*e.g.*, several routes to reach a destination), and the costs of performing the tasks may vary depending on the choice. If several robots use the same resource (reflected in a relationship between their choices), we allow interference between them to be modelled. Inter-agent interference is treated mathematically as a penalization to the cost of performing that task. In this manner, we can model shared resources and generalize the conventional MRTA problem formulation to include resource contention. The result is an optimization problem for finding the minimum-cost solution including the interference induced penalization cost. We term this the multiple-choice assignment problem with penalization (mAPwP).

In general, there are many ways penalization costs could be estimated. When evaluation of the interference is polynomial-time computable, we call this the mAPwP problem with polynomial-time computable penalization function (mAPwPP). Even with a cheaply computable penalization function, we show that mAPwPP is NP-hard. We also investigate two other problems that have particular forms of penalization functions: linear penalization functions and general convex functions. We show that the two problems are in P and NP-hard, respectively. We provide an exact algorithm and two polynomial-time algorithms for the problems. The algorithms are domain-independent so that it can be used for many multi-agent scenarios that have quantifiable interference between agents.

The remainder of this paper is organized as follows. Section II discusses the related literature on optimization methods for MRTA. Section III defines the problem mathematically, and Section IV describes the NP-hardness results. Section V presents algorithms, Section VI describes experiments, and the final section concludes.

Both authors are with the Department of Computer Science and Engineering at Texas A&M University, College Station, Texas, USA. cjnam,dshell at cse.tamu.edu

II. Related Work

The equivalence of the classical assignment problem by a network flow problem has been well known for decades. This may lead to the suggestion that one can prevent interference by imposing additional constraints in the form of capacity constraints in the flow formulation. This can be solved by a centralized manner [11] or a distributed manner [12]. However, that approach models interference as a binary penalization, which is zero or infinite, whereas incurred by resource contention are more widely applicable if the interference is modeled as a continuous function that increases proportionally to the amount of interference. (See, for example, our use of published and validated traffic models in Section VI.)

An alternative is for the mAPwPP can be cast as a linearly constrained 0-1 programming problem, with the penalization function incorporated into the objective function with the cost sum. The objective function is optimized over a polytope defined by the mutual exclusion and integral constraints. The results in this paper suggest that one can have an optimal solution in polynomial time if the penalization function is linear. When the penalization is more complex, a common method to solve the problem is enumeration, for example using the branch-and-bound method, but its time complexity in the worst case is as bad as that of an exhaustive search; rather more insight is gained by employing the method we introduce in this paper. Many practical algorithms [14], [15], [16] are suggested in the literature, but they also have exponential running time in the worst case. Linearizing the complex penalization function could be an alternative to have polynomial running time but has no performance guarantee.

III. Problem Description

A. Bipartite Multigraph

The mAPwP problem can be expressed as a bipartite multigraph. Let $G = (R, T, E)$ be a bipartite multigraph consisting of two independent sets of vertices R and T, where $|R| = n$ and $|T| = m$, and a collection of edges E. An edge is a set of two distinct vertices denoted (i, j) and incident to i and j. Each edge in G is incident to both a vertex in R and a vertex in T, and p_{ij} is the number of edges between two vertices. The vertices in R and T can be interpreted as n robots and m tasks, respectively. An edge is a way in which a robot may use resources, for which it expected to select one among p_{ij} choices for a given task. The precise interaction between resources is modelled via penalization function, described next.

B. Multiple-Choice Assignment Problem with Penalization (mAPwP)

Given n robots and m tasks, the robots should be allocated to tasks with the minimum cost. Each allocation of a robot to a task can be done via one of the p_{ij} choices where i and j are indices of the robots and the tasks, respectively. Each of the p_{ij} choices represents some set of resources used by a robot to achieve a task. The multiple choices indicate the resources can be used in many ways. We assume we are given c_{ijk}, the interference-free cost of the i-th robot performing the j-th task through the k-th choice. Let x_{ijk} be a binary variable that equals to 0 or 1, where $x_{ijk} = 1$ indicates that the i-th robot performs the j-th task in the k-th manner. Otherwise, $x_{ijk} = 0$.

In problem domains where multiple robots share resources, use of the same limited resource will typically incur a cost. We model this via a function which corrects the interference-free assignment cost (*i.e.*, the linear sum of costs) by including the additional cost of the effects of resource contention. We assume that the cost and the penalization are nonnegative real numbers. We also permit the cost to positive infinity when interference is catastrophic (or, for example, only one robot is permitted to use the resource). We assume $n = m$. If $n \neq m$, dummy robots or tasks would be inserted to make $n = m$. Then a mathematical description of the mAPwP problem is

$$\min \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p_{ij}} x_{ijk}c_{ijk} + Q(x_{111}, x_{112}, \ldots, x_{11p_{11}}, \ldots, x_{nmp_{nm}}), \tag{1}$$

subject to

$$\sum_{j=1}^{m}\sum_{k=1}^{p_{ij}} x_{ijk} = 1 \qquad \forall i, \tag{2}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{p_{ij}} x_{ijk} = 1 \qquad \forall j, \tag{3}$$

$$0 \leq x_{ijk} \leq 1 \qquad \forall\{i, j, k\}, \tag{4}$$

$$x_{ijk} \in \mathbb{Z}^{+} \qquad \forall\{i, j, k\}. \tag{5}$$

We note —without proof, owing to limited space— that (5) is superfluous because the constraint matrix satisfies the property of totally unimodular (TU) matrix.[1]

C. Penalization

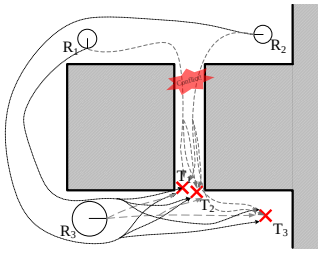
The penalization function maps a particular assignment to the additional cost associated with the interference. In the formulation of mAPwP earlier, $Q(\cdot)$ denotes the penalization function in most general terms. The mAPwP becomes the mAPwPP when $Q(\cdot)$ is computable in polynomial time.

The input domain for Q has $\sim O(\max\{n, m\}! \cdot (\max\{p_{ij}\})^{\min\{n,m\}})$ elements; in most cases a penalization function is more conveniently written in some factorized form. One example is if one is concerned only with the number of robots using a resource, not precisely the identities of the robots that are. If $Q_l(n_l)$ is the penalization function of the l-th choice where n_l is the number of robots for that choice, then the total penalization could be written as:

$$\begin{aligned} &Q(x_{111}, x_{112}, \ldots, x_{11p_{11}}, \ldots, x_{nmp_{nm}}) \\ &= Q_1(n_1) + Q_2(n_2) + \ldots + Q_q(n_q) \\ &= \sum_{l=1}^{q} Q_l(n_l). \end{aligned} \tag{6}$$

where q is the total number of choices in an environment. If the robots are homogeneous, n_l is the same as the number

[1]The standard treatment of the Optimal Assignment problem without a penalization factor for task allocation (*e.g.*, in [7]) considers only a bipartite graph (*i.e.*, $\forall_i \forall_j p_{ij} = 1$). Although TU is well-known for the problem, we believe this to be the first recognition of this fact for the problem above.



(a) An example of the mAPwP problem.

(b) A weighted bipartite multigraph.

Fig. 1: An example of the mAPwP problem. (a) Robots have multiple ways to reach their destinations. Interference occurs when both R_1 and R_2 use the narrow passage in the center. (b) A weighted bipartite multigraph representation of the example. (Weights are omitted for brevity.) An edge represents a resource (a passage) to perform a task (to reach a destination).

of robots on the l-th choice. Otherwise, each robot has a weight that represents the occupancy of the robot. If $Q(\cdot)$ is convex, the mAP becomes the mAP with convex penalization function (mAPwCP). Especially, it comes to be the mAP with linear penalization function (mAPwLP) if $Q(\cdot)$ is linear.

D. An Example

An example of the mAPwP is shown in Fig. 1(a). The goal is to minimize the total traveling time by distributing robots (R_1, R_2 and R_3) to three destinations (T_1, T_2 and T_3). R_1 and R_2 can use all the paths, but R_3 cannot use the narrow passage that T_1 and T_2 are located in because R_3 is wider than the passage. There will be interference if both R_1 and R_2 try to reach their destinations through the passage, so a time delay is incurred, which must be added to the total traveling time. A weighted bipartite multigraph that is equivalent to the example is shown in Fig. 1(b). The graph has $|R| = |T| = 3$ vertices, and every pair of vertices has 2 edges except for $p_{31} = p_{32} = p_{33} = 1$. In this example, $x_{111}, x_{121}, x_{131}, x_{211}, x_{221}$ and x_{231} are indicator variables reflecting use of the narrow passage, and interference occurs when more than one of these six variables are nonzero.

IV. NP-Hardness of mAPwP Problems

In this section, we show the mAPwPP and mAPwCP are NP-hard optimization problems, and the mAPwLP is in P. We prove the corresponding decision problem of the mAPwPP (D-mAPwPP) is NP-complete to prove the problem is an NP-hard optimization problem [17]. Then we briefly describe the mAPwLP is in P and show the mAPwCP is NP-hard.

A. The mAPwPP is NP-hard

Theorem 4.1 The D-mAPwPP problem is in NP.

Proof. The decision problem (D-mAPwPP) simply asks whether an assignment has cost less than a given threshold. Input: n robots, m tasks, p_{ij} choices, a polynomial-time computable penalization function Q, and costs of edges c_{ijk}, a constant α.
Question: Is the penalized cost of a given assignment less than α?
Certificate: An arbitrary assignment x_{ijk}.
Algorithm:

1 Check whether the assignment violates any constraints
2 Calculate the total cost of the assignment
3 Penalize the cost by the penalization function
4 Check whether the penalized cost is less than α

This is polynomial-time checkable so that D-mAPwPP problem is in NP. □

Theorem 4.2 The D-mAPwPP is NP-hard.

Claim. The proof is based on relation to the classic boolean satisfiability problem. The 3-CNF-SAT problem asks whether a given 3-CNF formula is satisfiable or not. It is a well-known NP-complete problem. If 3-CNF-SAT $\leq_{\mathrm{P}}$ D-mAPwPP, then D-mAPwPP is NP-hard.

Proof. The reduction algorithm begins with an instance of 3-CNF-SAT. Let $\Phi = C_1 \wedge C_2 \wedge ... \wedge C_k$ be a 3-CNF boolean formula with k clauses over n variables, and each clause has exactly three distinct literals. We shall construct an instance of the D-mAPwPP problem where $p_{ij} = 1$ ($i = 1, ..., n$ and $j = 1, ..., 2n$) such that Φ is satisfiable if and only if the solution of the instance of D-mAPwPP problem has cost less than a constant α.

We construct a bipartite multigraph $G = (R, T, E)$ as follows. We place n nodes $r_1, r_2, ..., r_n \in R$ for n variables and $2n$ nodes $t_1, f_1, t_2, f_2, ..., t_n, f_n \in T$ for truth values (true and false) of the variables. For $i = 1, ..., n$ and $j = 1, ..., 2n$, we put edges $(r_i, t_i) \in E$ and $(r_i, f_i) \in E$ where t_i and $f_i \in T$. The costs of the edges are given by c_{ij}. In addition, we construct an assignment by assigning vertex i in R to vertex j in T only when $x_{ij} = 1$ for $i = 1, ..., n$ and $j = 1, ..., 2n$. (Note that $x_{ij} \in \{0, 1\}$.)

Now, we construct a function Φ_{J} as follows. Each clause in Φ is transformed to a sum of terms in parentheses so that the terms correspond to the three literals in the clause. For a positive literal, we put x_{ij} where i is equal to the index of the literal and $j = 2i - 1$ whereas $j = 2i$ for a negative literal. Disjunctions of clauses are transformed to multiplications. A penalization of an assignment is defined as

$$Q = \begin{cases} 0 & \Phi_{\mathrm{J}} > 0 \\ \mathrm{N} & \text{otherwise,} \end{cases} \tag{7}$$

where N is a large number. If Φ_{J} has a solution which makes $\Phi_{\mathrm{J}} > 0$, the penalization is zero. Therefore, the cost of the assignment is $\sum_{i,j} c_{ij} x_{ij}$ and $Q = 0$ so the assignment has the total cost $\sum_{i,j} c_{ij} x_{ij}$. Otherwise, it will have a large nonzero penalization such as N. We can easily construct Q from Φ in polynomial time.

As an example, consider the construction if we have

$$\begin{aligned} \Phi = &(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee \neg x_5) \\ &\wedge (x_3 \vee \neg x_1 \vee \neg x_2), \end{aligned} \tag{8}$$

then the transformation is shown in Fig. 2. Φ has five variables so five nodes and ten nodes are placed in R and T, respectively. The nodes in R and T which have the same subscripts are connected. We produce function:

$$\begin{aligned} \Phi_{\mathrm{J}} = &(x_{11} + x_{23} + x_{48}) \cdot (x_{23} + x_{47} + x_{5 \cdot 10}) \\ &\cdot (x_{35} + x_{12} + x_{24}), \end{aligned} \tag{9}$$

and its penalization will be 0 or N depending on the assignment.

We show that this transformation is a reduction in a little more detail. First, suppose that Φ has a satisfying assignment.

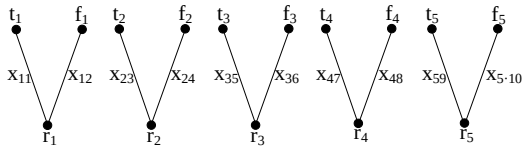


Fig. 2: The D-mAPwPP problem derived from the 3-CNF formula $\Phi = (x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_3 \vee \neg x_1 \vee \neg x_2)$. A satisfying assignment of Φ has $x_1 = 1, x_2 = 1, x_3 = 1$, and x_4, x_5 either 0 or 1. Corresponding assignment is that $x_{11} = 1, x_{12} = 0, x_{23} = 1, x_{24} = 0, x_{35} = 1, x_{36} = 0$. The values of other elements do not affect the satisfiability of Φ. This assignment makes $\Phi_{\mathrm{J}} > 0$.

Then each clause contains at least one literal that true is assigned, and each such literal corresponds to a matching of r_i and t_i. On the contrary, a literal assigned false corresponds to a matching of r_i and f_i. Thus, assigning truth values to the literals to make Φ satisfied yields matchings between R and T. We claim that the matchings are an assignment which makes $\Phi_{\mathrm{J}} > 0$. The assignment makes each sum of three terms (in parentheses) at least 1 so that Φ_{J}, a multiplication of the parenthesized terms, is greater than or equal to 1. Therefore, by the construction, we can get the total cost of the assignment and answer whether the cost is less than α.

Conversely, suppose that the D-mAPwPP problem has an assignment that makes $\Phi_{\mathrm{J}} > 0$. We can assign truth assignments to the literals corresponding to the matchings between R and T so that each clause has at least one variable which is true. Since each clause is satisfied, Φ is satisfied. Therefore, 3-CNF-SAT $\leq_{\mathrm{P}}$ D-mAPwPP. □

Corollary 4.3 By Theorem 4.1 and 4.2, the D-mAPwPP problem is NP-complete. Therefore, the mAPwPP problem is NP-hard optimization problem.

B. The mAPwLP is in P

Mathematically, the mAPwLP can be cast as an integer linear programming problem whose constraint matrix satisfies the property of totally unimodularity. This problem can be solved in polynomial time as described in [18, Corollary 2.2]. Therefore, the mAPwLP is in P.

C. The mAPwCP is NP-hard

The mAP with a convex quadratic penalization function (mAPwCQP) is a proper subset of the mAPwCP, and is a natural next step after examining mAPwLP. The mAPwCQP has the form

$$\min \{x^T H x + cx \colon Ax \leq b, x \in \{0,1\}\} \tag{10}$$

where H is positive semidefinite and symmetric, c is nonnegative, A is TU, and b is integer.

The following binary quadratic programming (BQP) is an NP-hard problem [19, Theorem 4.1]. The BQP problem is

$$\min \{y^T M y + dy \colon A'y \leq b', y \in \{0,1\}\} \tag{11}$$

where $M = L^T D L$, $D = I$, $d = 0$, L is TU and nonsingular, A' is TU, and b' is integer.

Theorem 4.4 The mAPwCQP is NP-hard.

Claim. If M is symmetric and positive semidefinite, we can reduce any BQP to an instance of the mAPwCQP. Namely, BQP $\leq_{\mathrm{P}}$ mAPwCQP.

Proof. Since $D = I$, $M = L^T L$. Then $(L^T L)^T = (L)^T (L^T)^T = (L^T L)$. Thus, M is symmetric.

For any column vector v, $v^T L^T L v = (Lv)^T Lv = (Lv) \cdot (Lv) \geq 0$. Thus, $L^T L$ is positive semidefinite. Therefore, BQP $\leq_{\mathrm{P}}$ mAPwCQP as we claimed. □

Lemma 4.5 mAPwCQP $\subsetneq$ mAPwCP.

Corollary 4.6 By Theorem 4.4 and Lemma 4.5, the mAPwCP is NP-hard.

V. Algorithms for mAP Problems

In this section, we devise algorithms for mAP problems. The exact algorithm for the mAPwPP recursively enumerates unpenalized assignments and their costs from the best assignment in terms of optimality, by calling a combinatorial optimization algorithm for each iteration. However, no enumeration and optimization algorithm exists for multigraphs, so we must extend Murty's ranking algorithm [20] and the Hungarian method [21] to the weighted bipartite multigraphs.

Then we suggest polynomial-time algorithms for the mAPwLP and the mAPwCP. For brevity, we denote them by the (optimal) mAPwLP algorithm and the (approximate) mAPwCP algorithm, respectively. The algorithms consist of two phases: the optimization phase and the rounding phase. In the first phase, we relax the integral constraint (5) so that a solution can be obtained in polynomial time, but it can be fractional. Thus, the second phase rounds a fractional solution to ensure the integrality of the assignment.

A. The Multiple-Choice Hungarian Method

We modify the labeling operations (the initialization and the update operations) and the path augmentation from the original Hungarian method. The labeling operations include all p_{ij} edges incident to i and j. In the path augmentation step, the minimum-weighted edge among p_{ij} is selected as the path between i and j. The pseudocode is given in Algorithm 1. The time complexity of this algorithm is $O(p^2(\max\{n,m\})^3)$.

Algorithm 1 The Multiple-Choice (MC) Hungarian method

Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph $G = (R, T, E)$ where $|R| = n, |T| = m$ and $p_{ij} = p, \forall\{i,j\}$.
Output: An optimal assignment M^* and its cost c^*.

1. Generate initial labeling $l(i) = \min_{1 \leq j \leq m}\{c_{ijk}\}, \forall i \in [1,n]$ and $l(j) = 0, \forall j \in [1,m]$ and matching M.
2. If M perfect, stop. Otherwise, pick an unmatched vertex $r \in R$. Set $A = \{r\}$, $B = \varnothing$.
3. If $N(A) = B$, update labels by
$$l(r) = l(r) - \delta \quad r \in A$$
$$l(t) = l(t) + \delta \quad t \in B$$
where $\delta = \max_{r \in R, t \in T-B}\{l(r) + l(t) + c_{ijk}\}$.
4. If $N(A) \neq B$, pick $t \in N(A) - B$.
   4a. If y unmatched, $u \to y$ is an augmenting path, then augment M and go to step 2.
   4b. If t is matched to z, extend alternating tree by $A = A \bigcup \{z\}$, $B = B \bigcup \{t\}$, and go to step 3.

Note: $N(r) = \{t | (r,t) \in G_{\mathrm{e}}\}$, where G_{e} is the equality graph, and $N(A) = \bigcup_{\forall r \in A} N(r)$.

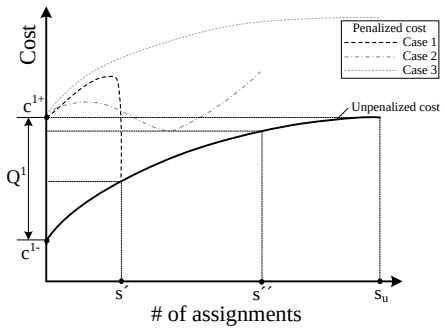


Fig. 3: All possible cases that the exact algorithm stops at s', s'' and s_{u}. s_{u} is the upper bound of the number of assignments to be enumerated to find an optimal solution.

B. The Multiple-Choice Murty's Ranking Algorithm

We modify the partitioning part of the original ranking algorithm. The set of all matchings is partitioned into subsets by removing each vertex and edges of s-th matching. After finding an optimal solution of each subset by Algorithm 1, the vertices and the edges of the optimal solution are recovered. In the removing and recovering procedures, p_{ij} edges are removed and recovered all together. The other parts are same as the original version. The time complexity of this algorithm is $O(sp^2(\max\{n,m\})^4)$.

C. Exact Algorithm for the mAPwPP

1) Algorithm Description: The pseudocode is given in Algorithm 2. We denote the s-th assignment before/after penalization as $X^{s-/+}$ and its cost is $c^{s-/+}$. Similarly, Q^s refers the penalization of the s-th assignment. In the first iteration (*i.e.*, $s = 1$), the algorithm computes the best assignment without penalization (c^{1-}). The penalization of the best assignment (Q^1) is computed and added to the cost of the best assignment ($c^{1+} = c^{1-} + Q^1$). Then, the algorithm computes the next-best assignment and compares its unpenalized cost (c^{s-}) with the minimum penalized cost to the previous step ($\min\{c^{1+}, ..., c^{(s-1)+}\}$). The MC Murty's ranking algorithm enables recursive computation of the next-best assignment (line 3). The algorithm repeats each iteration until either of the following conditions are met: when an unpenalized cost is greater or equal to the minimum penalized cost so that $\min\{c^{1+}, ..., c^{(s-1)+}\} \leq c^{s-}$, or the algorithm has enumerated all assignments ($N_\Pi = {}_mP_n \times \Pi_{i,j}^{n,m} p_{ij}$).

Fig. 3 shows three cases in which the algorithm stops. The algorithm enumerates, at most, s_{u} assignments, but may enumerate fewer if case 1 or 2 occur. Therefore, s_{u} is the upper bound of the number of assignments to be enumerated. We assume $p_{ij} = p, \forall\{i.j\}$ so that the number of choices is the same. If the p_{ij}s are not identical, then we may add dummy edges with infinite cost. The exact algorithm guarantees the optimality but has potentially impractical running-time, as it may enumerate factorial numbers (N_Π) of iterations in the worst case.

Algorithm 2 Exact algorithm

Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph $G = (R, T, E)$ where $|R| = n, |T| = m$ and $p_{ij} = p, \forall\{i, j\}$, and penalization functions Q_l for all l.
Output: An optimal assignment X^* and its cost c^*.

```
1  Initialize s = 1
2  while s < N_Π
3      Compute X^s and c^{s-} /* MC Murty's ranking algorithm */
4      if s = 1
5        Compute Q^s and c^{s+} = c^{s-} + Q^s
6        s = s + 1
7      else
8        if (c^{s-} ≥ min{c^{1+}, ..., c^{(s-1)+}})
9          X* = X^{s-1} and c* = min{c^{1+}, ..., c^{(s-1)+}}
10         return X*, c*
11       else
12         Compute Q^s and c^{s+} = c^{s-} + Q^s
13         s = s + 1
14       end if
15     end if
16 end while
17 X* = X^s and c* = min{c^{1+}, ..., c^{s+}}
18 return X*, c*
```

D. Optimal Algorithm for the mAPwLP

The first phase uses an IPM for linear programming (LP). LP has the optimal solution on a vertex of a polytope. All vertices of a polytope defined by a TU matrix are integer. However, an IPM may produce a fractional solution in which a problem has multiple optimal solutions [18]. In this case, all optimal solutions form an optimal face of the polytope [22]. It is then likely that an IPM converges to an interior point of this optimal face, which is not integer. In this case, we use the MC Hungarian method to choose one of the multiple optimal solutions which is integer. The pseudocode is not given due to the space limit but same with Algorithm 3 except Line 1: it uses an IPM for LP.

The time complexity of the IPM for LP that we used is $O((\max\{2n, nmp\})^3 L)$ [23][2] where L is the bit length of input variables. The Multiple-Choice Hungarian method has $O(p^2(\max\{n,m\})^3)$ complexity. Thus, the overall time complexity is $O((\max\{2n, nmp\})^3 L)$. We use MOSEK optimization toolbox for MATLAB [25], particulary `msklopt` function.

E. Approximation Algorithm for the mAPwCP

The pseudocode is given in Algorithm 3. The first phase uses an IPM for a convex optimization problem. The objective function must be twice differentiable to use the IPM. In convex programming, the solution could be fractional because not only are there multiple optimal solutions but also it is the unique optimal fractional solution. We also use the MC Hungarian method to round fractional solutions. However, the rounded solution may not be an optimal integer assignment. We provide its performance guarantee.

Theorem 5.1 The performance guarantee of the mAPwCP algorithm is $\max\{Q_{1,...,N_\Pi}\} - \min\{Q_{1,...,N_\Pi}\}$.

[2]The state of the art is [24] whose complexity is $O(\frac{\max(2n,nmp)^3}{\ln \max(2n,nmp)} L)$.

Proof. Let $P_{1,\ldots,N_\Pi}$ be assignments of an mAPwCP instance and $Q_{1,\ldots,N_\Pi}$ be the penalizations of the assignments. Without loss of generality, all $P_{1,\ldots,N_\Pi}$ have the same unpenalized cost. Let J be the largest integer solution among $P_{1,\ldots,N_{\Pi-1}}$ and P_{N_Π} be the optimal assignment whose cost is J^*. Then

$$J = \mathrm{K} + \max\{Q_{1,\ldots,N_{\Pi-1}}\}$$

and

$$J^* = \mathrm{K} + \epsilon + Q_{N_\Pi}.$$

Since $J \geq J^*$, $\max\{Q_{1,\ldots,N_{\Pi-1}}\} \geq Q_{N_\Pi}$ which means $Q_{N_\Pi} = \min\{Q_{1,\ldots,N_\Pi}\}$. Then

$$\begin{aligned} J - J^* &= J - (\mathrm{K} + \epsilon + Q_{N_\Pi}) = J - \mathrm{K} - \epsilon - Q_{N_\Pi} \\ &\leq J - \mathrm{K} - Q_{N_\Pi} = \max\{Q_{1,\ldots,N_{\Pi-1}}\} - Q_{N_\Pi} \\ &= \max\{Q_{1,\ldots,N_{\Pi-1}}\} - \min\{Q_{1,\ldots,N_\Pi}\}. \end{aligned}$$

Since $\max\{Q_{1,\ldots,N_\Pi}\} \geq \max\{Q_{1,\ldots,N_{\Pi-1}}\}$,

$$J - J^* \leq \max\{Q_{1,\ldots,N_\Pi}\} - \min\{Q_{1,\ldots,N_\Pi}\}. \qquad \square$$

Note that the performance guarantee is the difference between the maximum and the minimum penalization.

The time complexity of an IPM for a convex optimization problem is $O((\max\{2n, nmp\})^{3.5}L)$ [26]. Thus, the overall time complexity is $O((\max\{2n, nmp\})^{3.5}L)$. We use MOSEK `mskscopt` function for the optimization phase. Table I summarizes the problems and algorithms.

TABLE I: A summary of the problems and algorithms.

Problem		mAPwLP	mAPwCP
Objective function		Linear	Convex
Complexity class		P	NP-hard
Algorithm	Step I	Linear programming (IPM)	Convex optimization (IPM)
	Step II	The Multiple-Choice Hungarian method	
Overall complexity		$O((\max\{2n, nmp\})^3 L)$	$O((\max\{2n, nmp\}^{3.5}L)$
Performance guarantee		Optimal	$\max\{Q_{1,\ldots,N_\Pi}\}$ $-\min\{Q_{1,\ldots,N_\Pi}\}$

Algorithm 3 The mAPwCP algorithm

Input: An $n \times mp$ cost matrix which is equivalent to a weighted bipartite multigraph $G = (R, T, E)$ where $|R| = n, |T| = m$, $p_{ij} = p, \forall\{i, j\}$, and convex penalization functions Q_l for all l.
Output: An optimal assignment X^* and its cost c^*.

1 Compute $X^*_{\mathbb{R}^+}$ and $c^*_{\mathbb{R}^+}$ /* IPM for CP */
2 Compute $\hat{X}^*_{\mathbb{Z}^+}$ and $\hat{c}^*_{\mathbb{Z}^+}$ /* MC Hungarian method */
3 $X^* = \hat{X}^*_{\mathbb{Z}^+}$, $c^* = \hat{c}^*_{\mathbb{Z}^+}$
4 **return** X^*, c^*

VI. Experiments

We demonstrate that the exact algorithm works well and returns a result in reasonable time for practically sized cost matrices. The mAPwLP and the mAPwCP algorithms produce solutions quickly for even larger matrices. We implemented all the algorithms in MATLAB. The solution quality is measured by a ratio of an approximated solution to an optimal solution $\eta = \frac{c'^*}{c^*} \geq 1$. We assume that $n = m$ and $p_{ij} = p, \forall\{i.j\}$ for all the experiments. As we detail next, both randomly generated problem instances and instances based on real-world scenarios were used to validate the algorithms. Also, they are demonstrated with physical robots in small-scale experiments in our laboratory. First, we provide some detail on the particular penalization models used.

A. Penalization Functions

A penalization function models the interference incurred in a particular environment, and should consider the specific aspects of the robots and environment. Simple examples based on a factorization that adds costs as a function of the number of robots utilizing a resource, include models in the form of linear and an convex quadratic functions. Following the form in (6), let those penalization models for use of the l-th resource be

$$Q_l(n_l) = \begin{cases} \beta_{\mathrm{L}} n_l + \beta'_{\mathrm{L}} & n_l \geq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

and

$$Q_l(n_l) = \begin{cases} \beta_{\mathrm{C}} n_l^2 + \beta'_{\mathrm{C}} n_l + \beta''_{\mathrm{C}} & n_l \geq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where β_{L}, β'_{L}, β_{C}, β'_{C}, and β''_{C} are constants.

For the multi-vehicle routing scenario, we used the classic flow model developed by domain experts to quantify traffic congestion [27]. Many models have been proposed in the literature that compute a traffic speed v (m/s) according to traffic density ρ (vehicle/m). We let $\rho = n_l$ because we only consider an instantaneous assignment problem. We use an exponential model for our application that travel time (sec) is used as cost

$$Q_l(n_l) = \begin{cases} \frac{d_l}{v_{\mathrm{f}}\left[1 - \exp\left\{-\frac{\lambda}{v_{\mathrm{f}}}\left(\frac{1}{n_l} - \frac{1}{\rho_j}\right)\right\}\right]} & \rho_j \geq n_l \\ +\infty & \text{otherwise,} \end{cases} \tag{14}$$

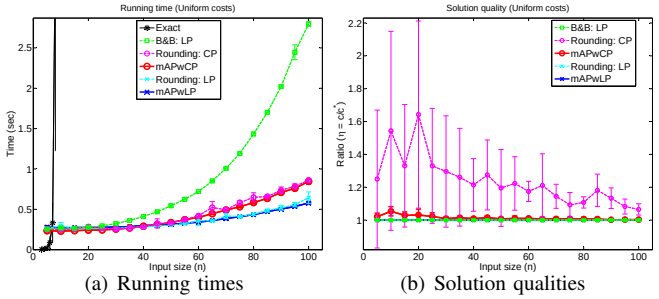
where v_{f} is the free flow speed (when $n_l = 0$), ρ_j is the jam density, and λ is the slope of the headway-speed curve[3] at $v = 0$, and d_l is the length of a resource that could be shared with other robots such as a passage.

B. Random Problem Instances

A uniform cost distribution ($\mathcal{U}(0, 60)$) is used to test the algorithms. The penalization function (13) is used for the exact and the mAPwCP algorithm, and (12) is used for the mAPwLP ($\beta_{\mathrm{L}} = \beta_{\mathrm{C}} = 1$ and $\beta'_{\mathrm{L}} = \beta'_{\mathrm{C}} = \beta''_{\mathrm{C}} = 0$). With fixed $p = 5$, the size of the cost matrix (n) increases from 5 to 100 at intervals of 5 (10 iterations for each n). Table II shows running times and solution qualities of our algorithms. We also compare them with conventional methods such as branch and bound (BB) and randomized rounding. We do not report results for the BB method on mAPwCP because the running-time is impractical and prohibitive even when the instance size is small (*e.g.*, $n = 10$). We display results in multiples of 10, owing to the space limitations.

The exact algorithm finds an optimal assignment in a reasonable time on small instances ($n \leq 8$); even a small problem has a huge search space (*e.g.*, $N_\Pi = 375,000$ when $n = 5$ and $p = 5$). The mAPwLP and the mAPwCP algorithms quickly find solutions even if n is large. Our methods are faster than the BB methods and similar to the randomized rounding methods. However, the methods we propose are the only to have polynomial running time. The solution qualities of the mAPwCP is better than the BB method (also the proposed algorithm has a performance guarantee).

[3]The ratio of (infinitesimal) velocity change over (infinitesimal) headway change.



(a) Running times (b) Solution qualities

Fig. 4: Running time and solution quality of random instance. (a) The mAPwLP and mAPwCP algorithms are slightly faster than the rounding method whose worst case running time is exponential. (b) The mAPwCP has better solution quality than the rounding method.

TABLE II: Running time and solution quality of random instances.

(a) The exact algorithm.

	n	3	4	5	6	7	8	9
Running time (sec)	Mean	0.0041	0.0115	0.0208	0.0894	0.3324	3.8327	95.1580
	Std. dev.	0.0043	0.0047	0.0144	0.0721	0.2467	2.6072	93.7848

(b) The mAPwLP and mAPwCP algorithms and existing methods.

		n	10	20	30	40	50	60	70	80	90	100
mAPwLP	Running time (sec)	Mean	0.2689	0.2743	0.2812	0.3003	0.3127	0.3406	0.3848	0.4333	0.5029	0.5786
		Std. dev.	0.0040	0.0091	0.0037	0.0064	0.0062	0.0046	0.0059	0.0081	0.0091	0.0067
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
mAPwCP	Running time (sec)	Mean	0.2296	0.2421	0.2546	0.2898	0.3354	0.4005	0.4908	0.5835	0.7094	0.8462
		Std. dev.	0.0051	0.0068	0.0055	0.0044	0.0071	0.0116	0.0101	0.0150	0.0187	0.0311
	Quality (η)	Mean η	1.0537	1.0314	1.0093	1.0092	1.0076	1.0104	1.0074	1.0067	1.0020	1.0030
		Std. dev.	0.0282	0.0353	0.0086	0.0078	0.0042	0.0105	0.0089	0.0100	0.0016	0.0020
B&B: LP	Running time (sec)	Mean	0.2688	0.2787	0.3235	0.4121	0.5442	0.7207	1.0080	1.4342	2.0219	2.7971
		Std. dev.	0.0160	0.0084	0.0040	0.0054	0.0162	0.0037	0.0070	0.0137	0.0080	0.0277
	Quality (η)	Mean η	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Rounding: LP	Running time (sec)	Mean	0.2972	0.2662	0.2644	0.2797	0.3051	0.3598	0.4173	0.4392	0.5186	0.6446
		Std. dev.	0.0386	0.0146	0.0072	0.0053	0.0067	0.0619	0.0305	0.0121	0.0295	0.0706
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Rounding: CP	Running time (sec)	Mean	0.2477	0.2841	0.2538	0.2816	0.3310	0.4203	0.4930	0.6485	0.7368	0.8602
		Std. dev.	0.0208	0.0266	0.0150	0.0041	0.0075	0.0164	0.0104	0.0564	0.0398	0.0249
	Quality (η)	Mean η	1.5435	1.6424	1.2960	1.2140	1.1957	1.1746	1.1448	1.1075	1.1332	1.0651
		Std. dev.	0.6057	0.5695	0.3391	0.1603	0.2342	0.0614	0.0731	0.0342	0.0630	0.0342

C. Multi-Vehicle Traffic Routing Problems

A multi-vehicle routing problem is used as a representative real-world application for our algorithms. We assume that n homogeneous robots and n tasks are distributed across p bridges in an urban area as shown in Fig. 6. The robots and the tasks are uniformly distributed within the boundaries. Distances from the robots to the tasks though the bridges are collected by using the Google Directions API [28]. The raw data are in meters (m) but converted to time (sec) according to v_f. Thus, the cost is travel time without congestion and penalized by the increased time owing to congestion. With fixed $p = 5$, n increases from 5 to 50 (3 to 9 for the exact algorithm). Other parameters are set as follows:

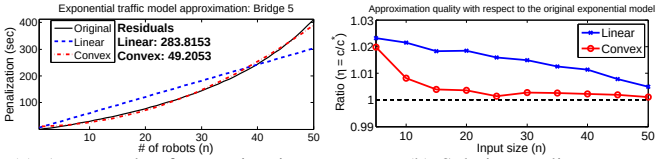
$$\begin{aligned} v_f &= 16.67 \text{ m/s}, \\ d_l &= \{500,\ 300,\ 250,\ 400,\ 200\} \text{ m}, \\ \rho_{jl} &= \{120,\ 80,\ 70,\ 90,\ 80\} \text{ robot/choice} \\ \lambda_l &= \{0.1389,\ 0.1667,\ 0.1528,\ 0.1944,\ 0.1389\} \text{ s}^{-1} \end{aligned}$$

where $l = 1, ..., 5$. The parameters reflect the characteristics of the real-world multi-vehicle routing problem.

We use (14) for the exact algorithm. However, our implementations for Algorithm 3 do not allow a complex exponential objective function like (14). Thus, we approximate (14) to a linear and a convex quadratic function such as (12) and (13). An example of the approximations is shown in Fig. 5(a). The quality of the approximation is measured by the sum of squared residuals. Table III shows the approximation results of all penalization functions of the five bridges.

TABLE III: Penalization function approximation results of the five bridges.

	Residuals				
Fitting type	Bridge 1	Bridge 2	Bridge 3	Bridge 4	Bridge 5
Linear	301.2	408.8	531.1	389.6	283.8
Convex	22.92	70.61	124.1	52.28	49.21



(a) An example of approximations (b) Solution quality

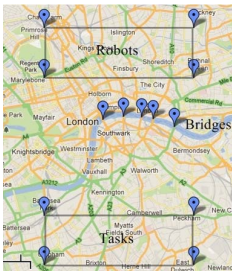
Fig. 5: (a) We approximate a complex exponential funtion to a linear and a convex quadratic function. (b) Solution qualities when the approximated functions are used for all five bridges.

The Fig. 5(b) shows solution qualities when the approximated functions are used. For each instance, we compute an optimal assignment by the exact algorithm when the original model is used. Then we compare it to the assignments when the approximated functions are used. As a result, the solution qualities are good (less than 1.024) so those approximations are acceptable.

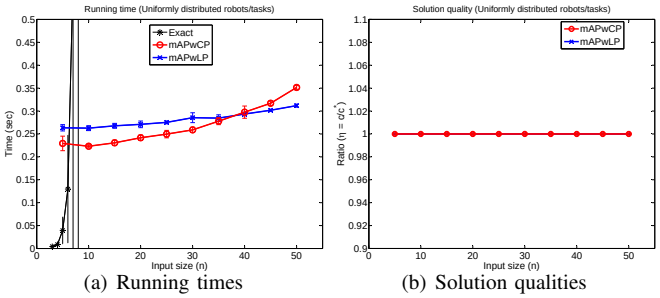
Fig. 7 and Table IV show the results (10 iterations for each n). The results are similar to the random instance case. This experiment shows that our algorithms can model realistic scenarios of robotic applications.

D. Physical Robot Experiment

We demonstrate that our method achieves global optimality even interference is not negligible. Fig. 8 shows the experimental setting. Two iRobot Creates (R_1 and R_2) have tasks of visiting the other robot's position on the opposite side of environment (T_1 and T_2). There are two passages to reach their destinations (shown as p_1 and p_2 in the figure). We use travel time as the cost and (14) as the penalization



6: Robots and tasks are located across five bridges. n robots and tasks are uniformly distributed in the upper and lower boxes, respectively.



(a) Running times (b) Solution qualities

Fig. 7: Running time and solution quality of the multi-vehicle routing problem. (a) The mAPwLP and mAPwCP algorithms quickly produce solutions. (b) The qualities are close to one for both algorithms.

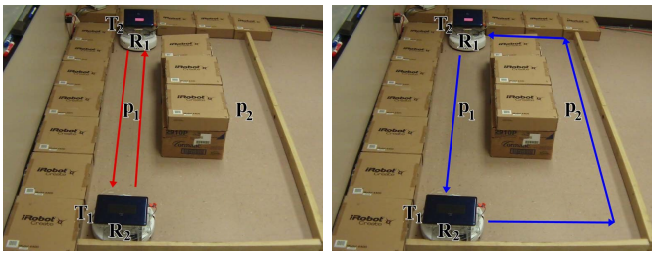
TABLE IV: Running time and solution quality of the multi-vehicle routing problem.

(a) The exact algorithm.

	n	3	4	5	6	7	8	9
Running time (sec)	Mean	0.0044	0.0088	0.0395	0.1298	0.5913	4.2897	126.0774
	Std. dev.	0.0015	0.0045	0.0290	0.1171	0.8772	6.8811	202.1757

(b) The mAPwLP and mAPwCP algorithms.

		n	5	10	15	20	25	30	35	40	45	50
mAPwLP	Running time (sec)	Mean	0.2634	0.2627	0.2678	0.2710	0.2753	0.2855	0.2846	0.2935	0.3017	0.3118
		Std. dev.	0.0069	0.0054	0.0054	0.0073	0.0032	0.0108	0.0072	0.0037	0.0024	0.0033
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
mAPwCP	Running time (sec)	Mean	0.2293	0.2232	0.2307	0.2417	0.2496	0.2589	0.2779	0.2976	0.3170	0.3515
		Std. dev.	0.0159	0.0034	0.0059	0.0058	0.0081	0.0056	0.0076	0.0133	0.0045	0.0040
	Quality (η)	Mean	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		Std. dev.	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000



(a) Both R_1 and R_2 use p_1. (b) R_1 uses p_1, and R_2 uses p_2.

Fig. 8: Two cases of resource use by two mobile robots. (a) Robots use the same resource so that interference is occurred. (b) Robots use different resources to avoid the interference.

function. We compute the assignment with Algorithm 2. We assume that R_1 and R_2 are identical. Space constraints and the data from the previous experiments forced us to omit reporting quantitative results.

When the robots move through the shortest path to the destination to attain the minimum travel time, they choose the same passage p_1 (Fig. 8a). However, this choice incurs interference between the robots. When the assignment is penalized, the best assignment is changed to the other assignment: R_1 uses p_1 and R_2 uses p_2 (Fig. 8b). When the robots use the same resource p_1, it takes 102 seconds to complete the tasks whereas the interference-free assignment takes 87 seconds.

VII. Conclusion

In this paper, we define the mAPwP problems and show that the mAPwPP, mAPwCP are NP-hard and mAPwLP is in P. We present an exact algorithm that generalizes Murty's ranking algorithm to solve the matching problem for weighted bipartite multigraphs, which employs the Hungarian method as a subroutine (which we also generalize to the multiple-choice case). In addition, we propose two polynomial-time algorithms for the mAPwLP and the mAPwCP. The mAPwLP algorithm produces an optimal assignment, and the mAPwCP algorithm computes a solution with bounded quality. In the experiments, we model interference among robots by introducing several penalization functions; the results show that the exact algorithm finds an optimal solution, and the mAPwLP and mAPwCP algorithms produce an optimal and a high-quality solution quickly. We also conduct physical robot experiments to show how resource contention aggravates optimality in practice and that the proposed algorithm achieves global optimality when an interference model is included.

References

[1] L. E. Parker, "Alliance: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics*, vol. 14, pp. 220–240, 1998.

[2] B. B. Werger and M. J. Mataric, "Broadcast of local eligibility for multi-target observation," *Distributed Autonomous Robotic Systems 4*, pp. 347–356, 2001.

[3] S. Botelho and R. Alami, "M+: a scheme for multirobot cooperation through negotiated task allocation and achievement," in *IEEE International Conference on Robotics and Automation (ICRA)*. Springer, 1999, pp. 1234–1239.

[4] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multi-robot coordination," *IEEE Transactions on Robotics*, vol. 18, pp. 758–768, 2002.

[5] M. B. Dias and T. Sandholm, "Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments," Carnegie Mellon University, Tech. Rep., 2004.

[6] D. Goldberg and M. J. Mataric, "Interference as a Tool for Designing and Evaluating Multi-Robot Controllers," in *Proceedings of the fourteenth AAAI National Conference on Artificial Intelligence (AAAI'97)*, Providence, RI, U.S.A., July 1997, pp. 637–642.

[7] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, pp. 939–954, 2004.

[8] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, "Multi-robot task-allocation through vacancy chains," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2003, pp. 2293–2298.

[9] J. Guerrero and G. Oliver, "Physical interference impact in multi-robot task allocation auction methods," in *Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. DIS 2006. IEEE Workshop on*. IEEE, 2006, pp. 19–24.

[10] H. Choi, L. Brunet, and J. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[11] T. C. Du, E. Y. Li, and A.-P. Chang, "Mobile agents in distributed network management," *Communications of the ACM*, vol. 46, no. 7, pp. 127–132, 2003.

[12] A. Kumar, B. Faltings, and A. Petcu, "Distributed constraint optimization with structured resource constraints," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 923–930.

[13] T. Matsui, H. Matsuo, M. Silaghi, K. Hirayama, and M. Yokoo, "Resource constrained distributed constraint optimization with virtual variables." in *AAAI*, 2008, pp. 120–125.

[14] H. W. Lenstra Jr, "Integer programming with a fixed number of variables," *Mathematics of operations research*, pp. 538–548, 1983.

[15] R. Kannan, "Minkowski's convex body theorem and integer programming," *Mathematics of operations research*, vol. 12, no. 3, pp. 415–440, 1987.

[16] D. Dadush, C. Peikert, and S. Vempala, "Enumerative lattice algorithms in any norm via m-ellipsoid coverings," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE, 2011, pp. 580–589.

[17] V. Kann, "On the approximability of np-complete optimization problems," Ph.D. dissertation, Royal Institute of Technology, 1992.

[18] T. K. Dey, A. N. Hirani, and B. Krishnamoorthy, "Optimal homologous cycles, total unimodularity, and linear programming," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 1026–1044, 2011.

[19] R. Baldick, "A unified approach to polynomially solvable cases of integer "non-separable" quadratic optimization," *Discrete Applied Mathematics*, vol. 61, no. 3, pp. 195–212, 1995.

[20] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations Research*, vol. 16, pp. 682–687, 1968.

[21] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[22] L.-H. Zhang, W. H. Yang, and L.-Z. Liao, "On an efficient implementation of the face algorithm for linear programming," *Journal of Computational Mathematics*, p. to appear.

[23] C. C. Gonzaga, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*. Springer, 1989.

[24] K. M. Anstreicher, "Linear programming in $o([\frac{n^3}{\ln n}]l)$ operations," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 803–812, 1999.
[25] A. Mosek, "The mosek optimization software version 6," *Online at http://www. mosek. com*, 2009.
[26] Y. Nesterov, A. S. Nemirovskii, and Y. Ye, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994, vol. 13.
[27] G. F. Newell, "Nonlinear effects in the dynamics of car following," *Operations Research*, vol. 9, no. 2, pp. 209–229, 1961.
[28] Google, "The Google Directions API," https://developers.google.com/maps/documentation/directions/, Jan. 2013.