Large-Scale Multi-Robot Task Allocation via Dynamic Partitioning and Distribution

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

This paper introduces an approach that scales assignment algorithms to large numbers of robots and tasks. It is especially suitable for dynamic task allocations since both task locality and sparsity can be effectively exploited. We observe that an assignment can be computed through coarsening and partitioning operations on the standard utility matrix via a set of mature partitioning techniques and programs. The algorithm mixes centralized and decentralized approaches dynamically at different scales to produce a fast, robust method that is accurate and scalable, and reduces both the global communication and unnecessary repeated computation. An allocation results by operating on each partition: either the steps are repeated recursively to refine the generalized assignment, or each sub-problem may be solved by an existing algorithm. The results suggest that only a minor sacrifice in solution quality is needed for significant gains in efficiency. The algorithm is validated using extensive simulation experiments and the results show advantages over the traditional optimal assignment algorithms.
Keywords: assignment partitioning, multi-robot task allocation, dynamic assignment.

1 INTRODUCTION

Task-allocation is a successful paradigm for coordinating multiple robots in a task-domain independent way. In its most straightforward form, each robot within the multi-robot team quantifies their expected individual performance on the pending tasks (their *utilities*). The robots then share this information and collectively allocate tasks among themselves so as to maximize estimated team performance, either directly or indirectly. It is not uncommon to perform the task assignment step repeatedly to ensure that the multi-robot system acts fluidly in a dynamic environment, deals with robot failures, takes newly injected tasks into account, and adapts as utility estimates are revised.

This paper is concerned with large-scale online assignment problems involving hundreds of robots and tasks. When one considers repeated allocation for large problems, it can be costly to use the naïve approach of recomputing by globally aggregating the latest utility estimates, and then performing a new allocation from scratch. Generally speaking, those steps are unavoidable when no information about the task structure is known. However, both spatial and temporal locality mean that after analyzing the utility matrix from a first task-assignment problem, subsequent reassignments—which typically involve quite similar inputs—may be greatly ameliorated. The approach we describe achieves its efficiency through parallelization of large allocation calculations and, additionally, communication costs are lessened by confining messages to subsets of the team.

The algorithm we introduce is based on the observation that the partitioning techniques for balancing matrices computations across multiple processors can not only be applied to distribute the computation loads but, in fact, application of these techniques may actually be directly used to compute an assignment problem. The approach computes the assignment solution in two stages. The first step involves a form of "abstraction" through which a coarse generalized assignment is done in which whole sets of robots are paired with sets of tasks. The associations between these sets are used to partition and distribute the assignment problems to smaller independent sub-assignments. Merging and splitting of the distributed sub-assignment problems allows online task commitment to be dynamic in response to changes in utility values. The partitioning and distribution steps essentially distill the global problem by eliminating insignificant utility information, reducing total computing and communication costs. In addition, the algorithm can permit one to tune the partitioning degree (or number of partitions) so that the level of decentralization is adjustable but can span the spectrum from fully centralized to entirely distributed. By comparing with centralized and decentralized algorithms, we show that this divide-and-conquer strategy possesses several advantages and that the new formulation as a partitioning problem has potential for further improvement via future research.

The contributions of this work include:

- Identification of a partitioning formulation for the assignment problem using hypergraph (and related matrix) representations, which enable a top-down, multi-level allocation of tasks.
- Demonstration that this leads to a new, naturally distributed solution in which centralized and decentralized aspects can be mixed and combined (up to any level of recursion for large scale problems).
- Detailed evaluation and exploitation of the sparseness properties of utility matrices.
- Introduction and solution of what we term the "dynamic" optimal assignment problem where random changes occur in elements of the utility matrix. The solution we describe naturally fits our hierarchical partitioning scheme

by ensuring that reallocation need only be propagated upwards to levels appropriate for adjustment of the assignment.

2 RELATED WORK

As described in Gerkey and Matarić [2004], task-allocation mechanisms are a specialized set of deliberative coordination approaches which focus on organizing and managing the performance of work performed by members of a multi-robot team. Several of these mechanisms were developed in conjunction with, and as foundational elements of, pioneering multi-robot software architectures. The focus of this paper is on the most widely studied (and most widely applied) variant of this problem, which Gerkey and Matarić term the Single-Robot-Tasks, Single-Task-Robots, involving an Instantaneous Assignment (ST-SR-IA).

Many variations and generalizations of the ST-SR-IA assignment problem have been developed: these include cases in which coalitions of robots must be formed [Berhault *et al.*, 2003], fine-grained sharing of resources [Tang and Parker, 2007], incremental assignment [Toroslu and Üçoluk, 2007], and sensitivity analysis [Liu and Shell, 2011]. Task-assignment problems of a similar underlying combinatorial nature have also been tackled from a hybrid systems perspective (*e.g.*, role assignment with preemption [Ji *et al.*, 2006], and potential-field hybrid controllers with relaxed mutual exclusion constraints [Zavlanos and Pappas, 2008]). An important recent result is that of Smith and Bullo [2009], who studied a class of problems in which assigned robots must remain at targets indefinitely, but for which important performance bounds under different environmental models could be identified.

A variety of algorithms exist for solving these problems, we distinguish two general classes: *Centralized* approaches involve a single decision-making agent that, after obtaining information about expected task utilities from the other robots, computes an assignment which it then broadcasts to the team. Most implementations of the primal-dual methods and linear programming-based approaches fall into this family, as do some auction protocols involving an auctioneer. Numerous examples are analyzed in Gerkey and Matarić [2004] and Dias *et al.* [2006]. *Decentralized* approaches do not distribute the utility information globally, instead individual agents may have little or no dependence on other robots. Some algebraic, greedy, market-based, and swarm-intelligence algorithms fall into this category (see *e.g.*, Dias *et al.* [2006], Kalra and Martinoli [2006], Kloder and Hutchinson [2006]). The taxonomy of Cao *et al.* [1997] further divides the decentralized framework into two types: *hierarchical* or *distributed*. Within hierarchical architectures, a certain degree of local centralization exists, while in distributed systems the robots are equal with respect to control, lacking any degree of centralization. For practical assignment strategies in distributed multi-robot systems, both Cao *et al.* [1997] and Dias *et al.* [2006] note that there are rarely completely centralized algorithms or purely distributed decentralized schemes, rather most methods are mixtures involving centralization and decentralization elements.

Centralized and decentralized architectures have advantages as well as their respective disadvantages. Fortunately they can be seen as complementary in some ways. Centralized approaches are relatively easy to implement and are frequently capable of computing an optimal result given a set of global information. Traditional centralized strategies require a central leader that is responsible for collecting, processing, and broadcasting the global information. Consequently, the communication complexity and computation complexity for the central leader can be a limiting factor that directly impacts the efficiency of the whole system. Additionally, this may result in a single point of failure. Decentralized methods, in contrast, achieve robustness to individual failure and may gain a performance advantage through parallel computation. Their biggest drawback lies in the realm of solution quality, especially when the system is highly decentralized so that significant information is never propagated throughout the system.

Although much research that is relevant to task-allocation has emerged in the last two decades, most work with a decentralized focus has a fundamentally static view: the structures which describe the degree of "local centralization" are fixed during the assignment process. Typically groups that operate tightly together are directly within the sensing/communication range (*e.g.*, Zavlanos *et al.* [2008], Lerman *et al.* [2006]), or are partitions or clusters in which roles and hierarchical level are predetermined (*e.g.*, Smith and Bullo [2009], Choi *et al.* [2009]). Some notable exceptions, which have a degree of philosophical similarity to the present work, are: (1.) Zlot and Stentz [2003] built a task tree after a task abstraction step, permitting a flexible assignment to be made at different levels; (2.) Simmons *et al.* [2002] extended the traditional three-layered architecture such that each layer can communicate directly with its peer layers on other robots, which allows the distributed robots to flexibly interact at multiple levels of abstraction with minimal communication overhead; (3.) The most recent work of Melo and Veloso [2011] also investigated the locality and interaction-sparsity, and showed that the local interaction of a decentralized multi-agent system can simplify the global coordination by introducing a decision-theoretic model called Dec-SIMDPs.

To date, the majority of decentralized/distributed assignment algorithms for multi-robot systems employ auction strategies (*cf.* Zavlanos *et al.* [2008], Michael *et al.* [2008], Goldberg *et al.* [2003], Dias *et al.* [2006], Bertsekas and Castanon [1991], Choi *et al.* [2009]). However, all these works do not decompose sub-assignments into independent partitions since they require either a complete communication network all the time or relaying capability among the distant robots [Zavlanos *et al.*, 2008; Choi *et al.*, 2009]. We are unaware of work that has modified the underlying auction mechanism in order to maximize flexibility of the assignment structure through dynamic merging and splitting of independent sub-assignments.

Different from above schemes, the approach we describe introduces a means for adapting to dynamic task changes within a hierarchical framework. The algorithm places an emphasis on a hierarchical decomposition derived from an initial analysis of the task structure, but maintains fluidity so that changes trigger readjustment and balancing operations are as localized as possible.

3 PROBLEM DESCRIPTION

Multi-robot task assignment $\mathcal{A}$ consists a set of robots R and a set of tasks T and can be denoted as $\mathcal{A} = \langle R, T \rangle$. In the ST-SR-IA assignment problem, following a given performance metric $P : R \times T \to \mathbb{R}^+$, the objective is to find an assignment solution $f : R \to T$ such that each robot $r_i \in R$ is assigned to a unique task $t_j \in T$, $r_i \mapsto t_j$ (i, j are indices of robots and tasks, respectively) so that either all robots are assigned or all tasks are allocated. In our work, $|R| \neq |T|$, the number of robots may differ from the number of tasks. The assignment of a robot to a task $r_i \mapsto t_j$ can be alternatively regarded as an allocation of a task to a robot $t_j \mapsto r_i$. Symmetry of injection and surjection, means one may assume $|R| \leq |T|$ without loss of generality, *i.e.,* we must ensure every robot will be assigned (injective but potentially not surjective). The multi-robot task assignment problem is to find a solution f such that the performance $\sum_{r \in R} P(r, f(r))$ is maximized.

Our desire is to partition the original assignment $\mathcal{A}$ into a serial of sub-assignments $A_1, A_2, \cdots, A_K$ satisfying:

$$\begin{aligned} & A_k = \langle R_k, T_k \rangle, \quad R_k \subseteq R, \quad T_k \subseteq T, \\ & \bigcup_{k=1}^{K} R_k = R, \ \text{and} \ \bigcup_{k=1}^{K} T_k = T, \\ & R_k \cap R_l = \varnothing, \ \text{and} \ T_k \cap T_l = \varnothing, \quad k, l \in [1, K], k \neq l. \end{aligned} \tag{1}$$

and all robots $r_i^{(k)\dagger} \in R_k$, will be only assigned to tasks $t_j^{(k)} \in T_k$ in the same partition. Therefore, after partitioning, we have $f_k : R_k \to T_k$, and the sub-assignments become *independent* and can be processed and distributed as fresh assignment problems. Finally, we want the aggregated performance of sub-assignments to have a small sacrifice of the original optimal performance. *i.e.*,

$$\sum_{r \in R} P(r, f(r)) \leq (1 + \epsilon) \sum_{k=1}^{K} \sum_{r^{(k)} \in R_k} P(r^{(k)}, f_k(r^{(k)})) \tag{2}$$

for some small ϵ value.

In order to better represent the robot–task relationship and the associated performance, it is instructive to consider both the matrix model and graph models, these are discussed in the following subsections.

3.1 Sparse Matrix Model

Following convention, the matrix describing estimates of robot performance when associated with a particular task is called the *utility matrix*: entry u_{ij}

† Superscript (k) denotes that this variable subjects to the k-th partition/sub-assignment. It applies to other variables throughout the paper.

represents the expected utility (or reward, or benefit) of having robot r_i perform task t_j. Since matrix transposition does not fundamentally change the problem, without loss of generality, we will consider utility matrices of size $m \times n$ ($m \leq n$). The linear sum assignment problem is to find the assignment that maximizes the sum of the accompanying utilities and, in our context, thereby maximizing the robot team's expected performance.

We use the term *shape* (m/n) to denote a matrix's height over width, and use *system size* ($m+n$) to describe how large the system is, not to be confused with the *matrix size* ($m \times n$). The matrix need not have every entry filled: expected utilities that are so small as to be unlikely to be part of the final assignment may be omitted—which we term void entries—producing a so-called *sparse* matrix. Such matrices enable considerable computational and communication savings, for example, when produced by having each robot transmit only non-negligible values during the initial utility aggregation procedure, or by a pre-processing step before computing the assignment.

A sparse utility matrix is particularly intuitive if the utility computation is dominated by factors that are a function of distance, which is often the case for mobile robots. When range limitations of physical sensors mean that data for utility estimation is only collected with acceptable accuracy for "nearby" tasks (*cf.* Kalra and Martinoli [2006]), then the sparsity of the utility matrix follows from the relative positions of robots and tasks. As demonstrated by the results in Section 6.1, spatial calculations mean that nearby information often contributes most significantly to the final task allocation solution. This is a useful property because estimates of utilities for far away tasks are also likely to be poorer than those nearby. Such distance effects may also naturally correlate with communication ability; in such cases partitioning (as described below) will not only reduce the total number of agent-to-agent messages but is also likely to localize them.

3.2 Hypergraph Model

It is well-known that an assignment can be described with a bipartite graph in seeking the optimal solution. Another graph representation that is useful and related to this work is called *hypergraph.*

Definition 1 *A hypergraph $\mathcal{H}$ is denoted by $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes (the counterpart of vertices), and $\mathcal{E}$ is a set of non-empty subsets of $\mathcal{V}$ called hyperedges. A hyperedge generalizes the notion of an edge in standard graphs, and instead connects an arbitrary set of nodes.*

The hypergraph representation is superior to a bipartite graph when considering assignment partitions because it has an interpretation in which the process can be intuitively understood as a clustering procedure. Viewing the nodes of the hypergraph as robots and the hyperedges as tasks, then the hypergraph forms a network in which properties of the utilities are reflected topologically. This is illustrated in Figure 1 where each hyperedge links to multiple nodes (robots) and is weighted by the associated utility.

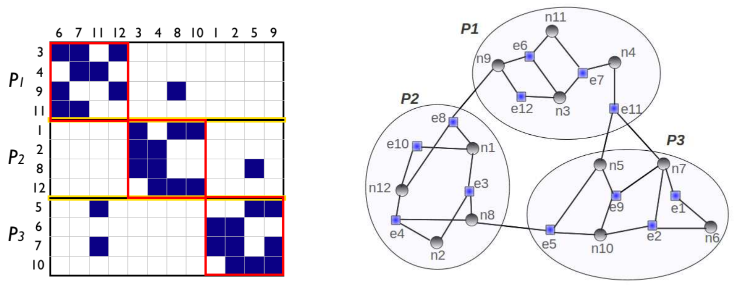
Graphs or matrices are isomorphic representations capturing the same underlying assignment problem in our work. Partitioning an assignment problem can be seen either in the graph formulation (bipartite graph or hypergraph), or directly in the utility matrix. However the different formulations lead to different interpretations for the solution finding process: the bipartite graph formulation involves searching for the minimal (weighted) edge cuts among sub-bipartite graphs (not necessarily complete graph with all vertex pairs edge-connected), while the hypergraph formulation involves shrinking hyperedges into disjoint singletons of maximal weight. We believe the partitioning approach introduced in this paper is most naturally viewed in this hypergraph form.

4 BACKGROUND: PARTITIONING

Graph partitioning is a general computational problem concerned with grouping vertices or nodes together so as to minimize the vertex/edge cuts between these groups. Many research areas benefit from graph partitioning, most relevant to this work is research on high performance computing. Graph partitioning is known to be NP-complete and consequently many heuristics have been proposed [Garey *et al.*, 1974]. *Spectral partitioning methods* and their variants (*e.g.* Dhillon [2001], Pothen *et al.* [1990], Hendrickson and Leland [1993]) have been used to partition a wide variety of graphs. With specialized strategies, particular properties (*e.g.*, imbalance among partitions) can be controlled. The Kernighan-Lin/Fiduccia-Mattheyses heuristic [Kernighan and Lin, 1970; Fiduccia and Mattheyse, 1982] performs well in refining partitions and converges quickly when a good initial partitioning can be provided. Geometric partitioning methods (*e.g.,* Miller *et al.* [1993] Gilbert *et al.* [1995]) are generally the fastest available but require multiple trials and the partitioning result is not deterministic. The widely used—and most relevant to this work—multilevel partitioning heuristic [Karypis and Kumar, 1998; Hendrickson and Leland, 1993] combines different heuristics in its three principle phases: *coarsening*, *initial partitioning* and *uncoarsening.*

Graph partitioning results in a pattern in the corresponding matrix representation: non-void entries of the sparse matrix are gathered into non-overlapping blocks on the diagonal by performing row/column swaps, such that the matrix is "partitioned" into clusters of void or non-void entries. This has been extensively researched for high performance computing applications [**?**; Kolda, 1998; Ümit V. Çatalyürek and Aykanat, 1999; Aykanat *et al.*, 2002; Arafeh *et al.*, 2008] where non-diagonal entries represent costly communications between processors. Total running time is reduced when (parallel) processor loads are balanced, *i.e.*, the diagonalized blocks are partitions of equal sizes. Communication volume is minimized when non-diagonal blocks have as few non-void entries as possible [Hendrickson *et al.*, 1998].

In this paper, the running time for the assignment problem is reduced by introducing parallelism analogous to the matrix partitioning treatment used more broadly in high performance computing. Size-balanced and independent


6 7 11 12 3 4 8 10 1 2 5 9
3 4 9 11 1 2 8 12 5 6 7 10
P1
P2
P3
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12
e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12

(a) (b)

Figure 1: (a) Partitioned sparse matrix (each shaded cell represents a non-void utility, and all non-void utilities are unit-weighted); (b) The corresponding hypergraph representation (circles are nodes and squares denote hyperedges).

diagonal blocks of the partitioned utility matrix represent sub-problems which can be distributed with minimal overhead. We distribute the assignment problem by partitioning the pre-processed utility matrix, which is accomplished by partitioning the equivalent hypergraph. A partitioning example showing the equivalence of the utility matrix and hypergraph representations is shown in Figure 1. The utility matrix is sparse since it has been processed so as to retain only a subset of the utilities that are most important, and the non-void entries are equally weighted (unit-weighted). As described below, this reduces the remaining subset to an equal preference set somewhat analogous to a greedy choice of a sub-set of tasks.

5 THE TWO-STAGE PARTITIONING AND DISTRIBUTION STRATEGY

The goal of our work is to partition and distribute the centralized assignment problem in order to address problems involving large numbers of robots and tasks, and to maximize responsiveness to task dynamics. The algorithm we describe arose by observing that changing row (robots) or column (task) ordering does not alter the outcome of the classical optimal assignment problem treatment (*e.g.*, [Kuhn, 1955]) for the multi-robot task allocation. The problem can be formalized in an integer linear programming form:

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} u_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., m, \\
& \sum_{i=1}^{m} x_{ij} \leq 1 \quad j = 1, ..., n, \\
& x_{ij} = 0 \text{ or } 1.
\end{aligned}
\tag{3}
$$

where $m \leq n$ in this case (every robot must be assigned, but some tasks may be redundant).

The computed assignment is a matrix X of entries x_{ij} which are zeros and ones. When $m = n$ it is most easy to see that this is a permutation matrix: post-multiplying the utility matrix by X^{T} reorders the tasks and the value of the optimal assignment is merely the trace of the result. Thus, a matrix reordering which maximizes mass along the diagonal represents the optimal assignment. This view is useful because it leads to the interpretation of an incremental assignment process as producing a sequence of increasingly block-diagonalized utility matrices that each represent selection of a subset of robots as candidates for a subset of tasks.

Proceeding along these lines, our approach uses a two-stage assignment strategy. In the first stage, aggregated assignment data are partitioned into K sub-assignment problems, in which the robot-task pairs are strongly "connected" and are likely to be assigned within the same partition. In the second stage, the K sub-assignments are regarded as K new independent assignment problems and the responsibility for solving each sub-problem is delegated to robots within the respective partitions. This permits the assignment to be computed in a distributed manner and in parallel. The approach is a multi-level strategy because each of the sub-problems can either be solved by applying the same procedure recursively, or by directly employing a classical assignment algorithm.

5.1 Matrix Sparsity Control

Our implementation employs a pre-processing step in order to control the degree of sparsity in the utility matrix: a sparsity parameter $\rho \in (0, 1]$ is used by a single robot which, after it has aggregated all available utility information, removes the $(1-\rho) \cdot n$ smallest elements from each row. (An alternative method is to remove those entries less than some threshold, but this requires that utilities have an absolute scale rather than a relative interpretation.) This filtering retains only the highest weighted entries, which are most likely to contribute to the assignment solution and produces a sparse matrix consistent with the format required for matrix factorization. Since negligible utilities are not needed by the algorithm, the utility aggregation can be simplified to only gather the highest weighted values. Significant computation can also be avoided in calculating utilities, for example, if an admissible heuristic [Russell and Norvig, 2009] is employed during planning, a threshold permits early termination for especially costly tasks.

The removal of any utility values may adversely affect the quality of the final assignment solution: indeed it is possible to contrive examples in which arbitrarily small utility values are necessary to produce the optimal assignment. In practice (and as demonstrated in Section 6) the utility matrices that arise in actual robot task-allocation problems permit a large proportion of the utility values to be discarded before a significant reduction in solution quality occurs. By manipulating ρ the quality-vs.-efficiency trade-off may be tuned to suit user needs; we believe this to be more representative of our physical robots than directly relating task sparsity to some fixed range (*e.g.,* based on communication radius, although *cf.* [Smith and Bullo, 2009] for such a treatment).

5.2 Determining the Sparsity Parameter ρ

We evaluate the solutions found through a decentralized assignment by comparing it with the global optimum, and quantify the quality of the solution using a *quality* parameter $q \in [0\%, 100\%]$, *e.g.*, a solution with $q = 50\%$ means it reaches 50% of the optimum for a maximization problem. Higher solution quality, in general, limits the degree to which a matrix may be filtered, *i.e.,* requiring a larger ρ. We use $\rho(q)$ to denote the threshold sparsity to reach a given quality q. Moreover, for a fixed q, $\rho(q)$ varies with the size of a system, as well as the shape of the matrix. However, our empirical results (presented in Section 6) show that, for a given matrix with known shape, even if the system size varies, few utilities in each row contribute most to the optimum and the number is approximately constant. This observation is very useful especially for the square matrix case, which is the scenario most described in the literature. For non-square matrices of size $m \times n$, the rule still holds and thus $\rho(q)$ can be estimated by running calibration experiments off-line, which is also discussed later.

5.3 Determining the Partitioning Number K

The degree to which the system is partitioned and the sizes of those partitions will affect the system's performance. Since balanced partitions are preferred, we can expect the final partitioned diagonal blocks to have similar sizes and be similar in shape. Thus, the question becomes one of determining the partitioning number K. Or, more generally, assessing a range of values for K which have attractive performance.

Complete utility matrix: If the utilities from all robot–task pairs are available or inexpensive to obtain, we will be able to obtain a complete dense utility matrix. Once ρ is determined, each row will have ρn non-void utilities after filtering. In order to cluster all non-void utilities into the diagonal blocks and leave the off-diagonal blocks containing as few non-void utilities as possible, one must ensure that $\rho n \leq n/K$, *i.e.*,

$$1 \leq K \leq \frac{n}{\rho n} = \frac{1}{\rho}. \tag{4}$$

Partial utility matrix: There are occasions when all utilities cannot be obtained directly, *e.g.*, some robots may be unaware of some of the tasks, or may be unable to compute utilities with acceptable accuracy for all the tasks. For the rows that contain a number greater than ρn non-void entries, the filtering procedure will operate as described in the sparsity control case, otherwise, all existing entries are retained. Let h_i denote the number of non-void utilities in row i (for the i-th robot), then we can approximately estimate the range of K via following formula:

$$1 \leq K \leq \frac{mn}{\sum_{i=1}^{m} \left(h_i I_{\{h_i \leq \rho n\}} + \rho n I_{\{h_i > \rho n\}} \right)}, \tag{5}$$

where I is an indicator variable that is equal to 1 if the condition is true and 0 otherwise.

5.4 Matrix Partitioning and Distribution

As in parallel computing applications, balanced partitions are preferred so that the computational workload may be equalized. The resultant K-partitioned matrix forms blocks containing approximately m/K rows and n/K columns, and the non-void blocks finally lie on the diagonal after simple re-ordering. Let U and B_k ($k \in [1, K]$) denote the sparse utility matrix and the partitioned diagonal blocks, then ideally we have $U = diag(B_1, B_2, \cdots, B_K)$ where B_k has dimension $m^{(k)} \times n^{(k)}$, and $\sum_{k=1}^{K} m^{(k)} = m$, $\sum_{k=1}^{K} n^{(k)} = n$. Thus

$$\begin{aligned} \sum_{j^{(k)}=1}^{n^{(k)}} x_{i^{(k)} j^{(k)}} &= \sum_{j=1}^{n} x_{i^{(k)} j} = 1 \quad \forall k,\ i^{(k)} = 1, ..., m^{(k)}, \\ \sum_{i^{(k)}=1}^{m^{(k)}} x_{i^{(k)} j^{(k)}} &= \sum_{i=1}^{m} x_{i j^{(k)}} \leq 1 \quad \forall k,\ j^{(k)} = 1, ..., n^{(k)}, \end{aligned} \tag{6}$$

where $x_{i^{(k)} j^{(k)}}$, $x_{i^{(k)} j}$ and $x_{i j^{(k)}}$ are the binary variables defined analogously to x_{ij} but relevant to diagonal block B_k. This shows that the assignment solutions from partitioned blocks do not violate the assignment constraints and collectively form a global assignment solution.

We call the 1st-level partitioned matrix the *assignment table*. In the assignment table, m rows are partitioned into K *belts* (see Figure 2(a)), with each belt having $K - 1$ void blocks and a single non-void diagonal block. In each partition, we randomly pick a robot as the *sub-leader*, *i.e.,* there are K sub-leaders in total. The initial assignment table is distributed to the K sub-leaders, and the sub-leaders are responsible for solving the independent assignment problems in their respective partitions. Figure 2(a) illustrates the assignment table with three belts that are distributed to three sub-leaders ($SL1, SL2$ and $SL3$) within their respective partitions. Note that although each sub-leader obtains a copy of the global assignment table, it may only manipulate the belt to which it belongs. However, the entire table provides global observations of the whole assignment problem, and a sub-leader combines it local observations with global information to determine whether merging or splitting of independent sub-assignments is required and which sub-leader(s) it should contact for such operations.

5.5 Dynamic Assignment

An initial partitioning can be of significant advantage in solving the assignment problem when task dynamics mean that reassignments are frequently necessary. This is quite common in online assignment problems. Once an initial partitioning has been constructed, it is not necessary to do the whole partitioning procedure for each reassignment query but, instead, one may localize the impact of changes in utility values. Computation should focus only on addressing those partitions which have changed significantly in a way that undermines the acceptability of the initial partitioning. Partitions operate in an entirely distributed fashion and continue to be considered independent of each other until sufficient utility values have "diffused" to entries outside the block-diagonal. Those values reflect tasks that might be currently assigned to robots in other partitions. When this occurs, a repartitioning is called for, but it need only be repeated among those interacting partitions.

To assess whether repartitioning is necessary, the sub-leaders monitor the sparsity of associated utility blocks in their respective partitioned belts. As mentioned in Section 5.2, only a few utilities contribute to the overall score, and this is sensitive only to the matrix shape but not the system size. Since the partitions are generally balanced, the resultant blocks have shape similar to original matrix, and for a block B with row size m_B and column size n_B, we have

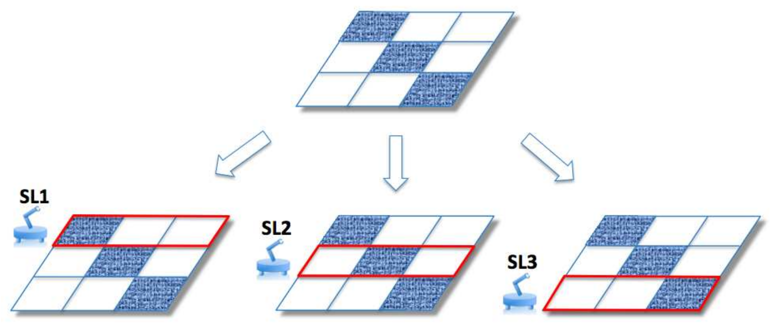
$$n_B \cdot \rho_B(q) = n \cdot \rho(q) \;\Rightarrow\; \rho_B(q) = \frac{n}{n_B}\rho(q), \tag{7}$$

where $\rho_B(q)$ is the threshold we use in block B to obtain an online assignment quality greater than or equal to the specified quality q. During a dynamic assignment, the utility values may change and the observed sparsity for a specific block is likely to vary too. Observed sparsity is measured by

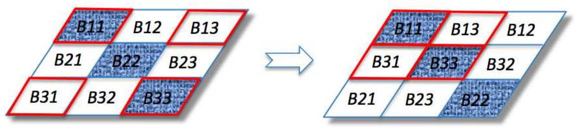
$$\hat{\rho}_B = \frac{1}{m_B n_B} \sum_{i=1}^{m_B} \sum_{j=1}^{n_B} I_{\{v_{ij}>0\}}, \tag{8}$$

where v_{ij} is the value of entry (i, j) in block B.

We compare the observed sparsity $\hat{\rho}_B$ of a given diagonal block B with the threshold $\rho_B(q)$. If $\hat{\rho}_B < \rho_B(q)$, it means that the robots or tasks are less suited to the current partition than they once were and are probably candidates for



(a)



(b)

Figure 2: (a) An assignment table with three partitioned diagonal blocks is distributed to three sub-leaders ($SL1$–3). The red rectangles illustrate the belts that are the responsibilities of respective sub-leaders; (b) An assignment table with interrelated blocks highlighted in red. After row and column permutations, interrelated blocks are “clustered” into an interrelated submatrix which can be repartitioned again.

other partitions. Put another way, the association between some agent–task pairs in this partition has weakened and the quality of this sub-assignment has deteriorated. Simultaneously, we can find some off-diagonal block(s) that have increased non-void entries. The blocks involved in such an interaction are the *interrelated* blocks, illustrated with the highlighted outline in the left of Figure 2(b). Interrelated blocks are blocks which:

1. satisfy $\hat{\rho}_B < \rho_B(q)$ if they are diagonal blocks;
2. are non-diagonal blocks that reside in the same rows as the diffusing diagonal blocks, and also possess the largest density increment in their rows;
3. are complementary blocks to those from 1 and 2, such that all of them eventually form a rectangle with original diagonal blocks still on the diagonal.

If we treat each block as an entry, then the interrelated blocks can "merge" to an inner *interrelated submatrix* embedded in the utility matrix as illustrated on the right of Figure 2(b). This is because diagonal blocks can be re-positioned with a few symmetric row and column permutations.

Once repartitioning is triggered, the sub-leaders of the interrelated blocks only communicate with each other and carry out the repartitioning work following our proposed 2-stage assignment procedure. If no repartitioning is required, the sub-leaders in independent partitions simply compute the assignment solution either by using a centralized algorithm, or through recursive application of the procedure to sub-sub-leaders, *etc.*

Finally, to guarantee the accuracy of global assignment, a fresh global new partitioning should be carried out after some period of time. This is because the extracted coarse features of initial global assignment can gradually lose accuracy as many iterations of interrelated blocks coalesce and are repartitioned. We trigger "restarts" by counting the number of repartition operations. If any of the distributed sub-leaders' counters reaches a predefined threshold number N_r, this sub-leader automatically takes the role of the global leader and re-aggregates global information as well as partitioning the assignment from scratch. The threshold N_r is an empirical value depending on the frequency of changes within the system, but it could also be updated dynamically during an online learning process.

5.6 Assignment Partitioning and Distribution Algorithm

Details from the previous subsections are captured in Algorithm 1. In the pseudo-code lines 1–4 describe the matrix pre-processing, partitioning, and distribution by a global leader. The remaining lines are executed by sub-leaders in a distributed fashion.

Lines 13–16 and lines 17–21 describe the merging and splitting of sub-assignments, which is like a dynamic divide-and-conquer strategy for manipulating the local centralization and local decentralization. Lines 17–21 can also

Algorithm 1 Assignment Partitioning and Distribution

```
Input: utility matrix $U$ (size $m \times n$), assignment quality $q$, repartitioning maximal number $N_r$
Output: decentralized assignment solution
    {/* global leader do following four steps */};
 1: determine sparsity parameter $\rho(q)$ and partitioning number $K$;
 2: filter out and keep $\rho(q) \cdot n$ largest utilities per row;
 3: make $K$ partitions, obtain assignment table with $diag(B_1, B_2 \cdots B_K)$;
 4: distribute assignment table to each partition;
    {/* sub-leaders ($SL$) do "for each" below in distributed fashion*/};
 5: for each partition (associated with $B$) parallel do
 6:   if $SL$ counts $\#repartitions > N_r$ then
 7:     $SL$ becomes global leader;
 8:     goto 1: ;
 9:   end if
10:   determine $\rho_B(q)$;
11:   for every update (at fixed frequency) do
12:     compute observed $\hat{\rho}_B$;
13:     if $\hat{\rho}_B < \rho_B(q)$ then
14:       locate and communicate among interrelated blocks $B_1^{'}, B_2^{'}, \cdots B_l^{'}$;
15:       merge interrelated blocks $B = B \cup \{B_1^{'}, B_2^{'}, \cdots B_l^{'}\}$;
16:     end if
17:     if $size(B) > size(U)/K$ then
18:       repartition $B$ into unit blocks;
19:       distribute the new partitions to new selected sub-leaders;
20:       update assignment table and $\#repartitions$;
21:     end if
22:     implement reassignment locally using either Algorithm 1 recursively or
        by employing a centralized allocation mechanism.
23:   end for
24: end for
```

be modified using recursive mechanism of Algorithm 1 itself. In addition, the partitioning number K determines the hierarchical level of decentralization in the spectrum. Two extreme cases are: if $K = 1$, the approach is completely centralized; if $K = m$, each robot forms an independent partition so that the structure is fully distributed.

However, the description of the algorithm omits two details which must be considered for an implementation: (1.) the choice of partitioning software; (2.) the criteria for selecting whether to recurse on a sub-case or solve using centralized allocation mechanism. These are discussed next.

As emphasized in Section 4, a significant body of work exists to address the partitioning problem. We have tested ~10 popular graph/matrix partitioning tools that are available on the web, and found that `hMeTis` [Karypis and Kumar, 1998] and `PaToH` [Ümit V. Çatalyürek and Aykanat, 1999] perform the best. (Actually `hMeTis` is slightly faster than `PaToH`). It is worth noting that our proposed assignment strategy itself—along with the partitioning implementation—includes aspects of the multi-level framework. The first stage of assignment, which yields K partitions, is a *coarsening* phase that collapses the strongly connected agent-task pairs into super nodes and thereby capturing essential features of the global information. The *initial partitioning* phase is trivial in our algorithm since the particular diagonal blocks are non-overlapped, so the partitions are independent and already identified. The second stage of our algorithm, which computes the assignment solution for each robot, corresponds to an *uncoarsening* phase that refines the final results. As already pointed out, this multi-level partitioning strategy has been broadly used in many partitioning algorithms and much software, and the advantages of this framework are discussed by several authors [Karypis and Kumar, 1998; Hendrickson *et al.*, 1998; Papa and Markov, 2007].

In our implementation, we opted to have a single partitioning phase with the second stage operation that always employed a centralized allocation algorithm. There are two distinct reasons: (i.) some partitioning strategies/tools (*e.g.*, the `hMeTis` and `PaToH`) are already designed using a bisection mechanism, and these programs utilize such bisection to recursively partition a graph into smaller pieces until they reach the designated size; (ii.) our implementation was primarily used for evaluation purposes; a single partitioning phase allows one to assess the effectiveness of the distributed computation most easily. For extremely large problem instances deep recursion may be the only viable solution. The primary design criteria for whether recursive subdivision is worth conducting is the cost of the management that is required (keeping track of sub-sub-leading agents, for example), and how naturally the problem instance can be distributed. This latter property is partially a function of the matrix block's sparsity, density, and the interrelationships between entries. Moreover, since the utility matrix does not necessarily need to be square, the allocation of more than one task to each robot also works with this multi-level partitioning and distribution framework by simply treating the ultimate partitioned units as new assignment problems.

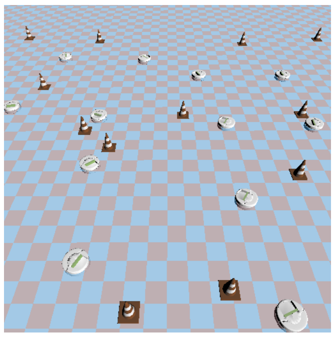
6 EXPERIMENTS

We simulated our algorithm by considering the problem of dispatching a group of robots to a set of destinations. This problem and variations on it has been employed for evaluating allocation strategies in the literature and forms a standard test problem (see, for example, Berhault *et al.* [2003], Liu and Shell [2011]). In order to integrate several popular open-source partitioning tools, we wrote a custom simulator in C++ and ran it in a GNU/Linux environment. Homogeneous robots begin from random positions within a $100m \times 100m$ square; they are provided with their position information and are given target locations (randomly positioned in the environment), as an example shows in Figure 3(a). The objective is to minimize the distance travelled by all of the robots. Dynamic scenarios are modelled by associating different drift speeds with both robots and tasks. The corresponding 2D environment for dispatching the large-scale multi-robot task allocation problems is shown in Figure 3(b).

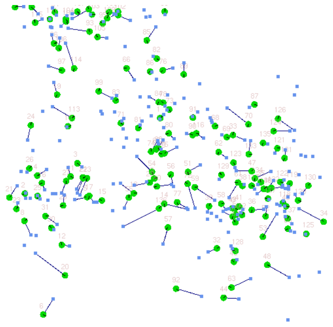
To adapt for our current optimal assignment software, we transformed the minimization problem to maximization problem by converting d to $-d$, where d is the real distance between a robot-task pair. We employed `hMeTis` [Karypis and Kumar, 1998]. In order to obtain blocks, we transpose the row-wise partitioned matrix and do a second partitioning on it. The diagonal block matrix is then obtained by simple block diagonalization.

6.1 Sparsity Analysis

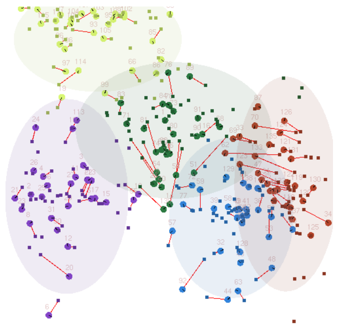
In this work we filter out small utilities that are likely to be dominated; first we need to investigate the effect that discarding this information has on the final assignment quality. We define metric $\eta_{x:y} = \frac{x}{y}$ as a measure of the practical performance of x over y. In this experiment, the quality q of an assignment solution is therefore equivalent to $\eta_{o:f}$ (o: optimum from *original* dense matrix; f:optimum from *filtered* sparse matrix). We investigated the matrices in different shapes (*i.e.*, manipulate m/n), and for each shape, we tested with different sizes (*i.e.*, control $m + n$). For each matrix of certain shape and size, the quality q under a serial of $\rho \in (0, 1]$ are observed. Figure 4(a) and 4(b) show the results of square matrices and rectangular matrices in fixed shapes. We are most interested in the sparsity threshold $\rho(q)$ that guarantees certain solution quality q. One interesting result is that, for each robot, although $\rho(q)$ varies along with the change of the number of tasks n, the product of them tends to be a constant number $N(q)$, *i.e.*, $N(q) = n\rho(q)$. In other words, the utility values that contribute most to the optimum assignment is only sensitive to the matrix shape, but not the system size. For instance, in Figure 4(a), when the matrix shape is fixed ($m/n = 1$ in this case), no matter what the system size is, the number of utilities that guarantees 95% solution accuracy is around 10, meaning that only $\sim$10 nearest tasks for each robot contribute significantly in determining the optimum. Comparing Figure 4(b) with 4(a) we see that the more slender the matrix (with greater tasks than robots), the smaller the $N(q)$ that is permissible. Once $N(q)$ is observed, $\rho(q)$ can be approximated via



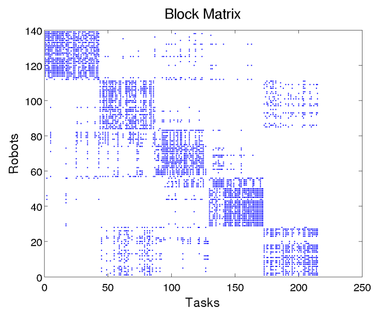
(a)



(b)

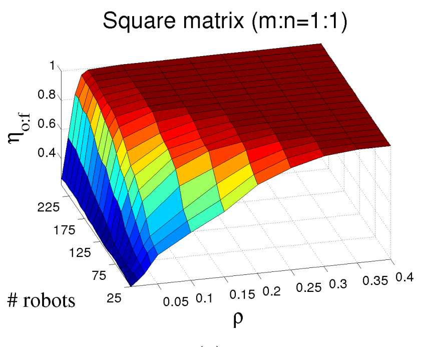


(c)

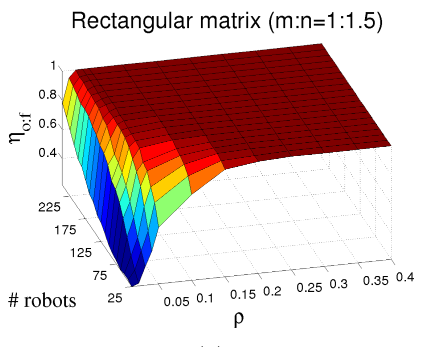


(d)

Figure 3: (a) An assignment problem to dispatch robots to nearest task locations; (b) Centralized assignment result in a 2D environment (circles are robots and square dots are tasks locations, lines between the robots and tasks represent the assignment); (c) Distributed assignment solution based on the partitioning results (circled areas denote partitions); (d) Block matrix of the corresponding partitions.



(a)



(b)

Figure 4: Sparsity analysis showing that significant proportions of the task matrix can be discarded, with little deterioration of assignment quality.

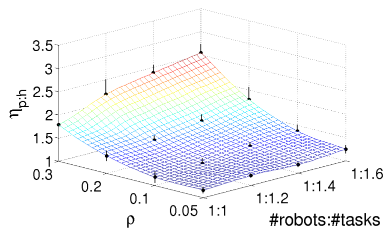
$\rho(q) \approx N(q)/n$.

6.2 Assignment Partitioning for the Static Case

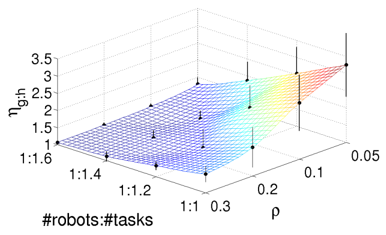
We have outlined in Section 5, the sparse matrix is converted into the format of hypergraph, and fed to `hMeTiS`. It produces a block matrix that provides information on the correspondence of robots and tasks in a way that causes them to be partitioned into clusters. An arbitrary assignment with partitioning results is shown in Figure 3(c) and 3(d). In Figure 3(c), there are 5 partitions obtained from the 5 blocks on the diagonal in the partitioned matrix, illustrated in Figure 3(d). Robots and tasks in the same partitions/blocks are considered as new assignment problems independent of others.

The second level assignment is executed inside each partition/block. In our work, we use the Hungarian algorithm [Kuhn, 1955] to solve the second level partitioned assignments. Note that it is possible that even voided (removed) entries will be assigned, and in the same row/column there could be many voided entries that have zero utility. The problem lies in that the assigned entry is not the smallest in reality (the real utility before being removed), which may deteriorate the optima considerably when ρ is too small. To solve this, we check such assigned pairs to avoid treating them as random allocations: if such assignments are found to be invalid, we re-adjust and thereby improve these results by greedily reassigning these robots to their nearest available (non-voided) tasks.

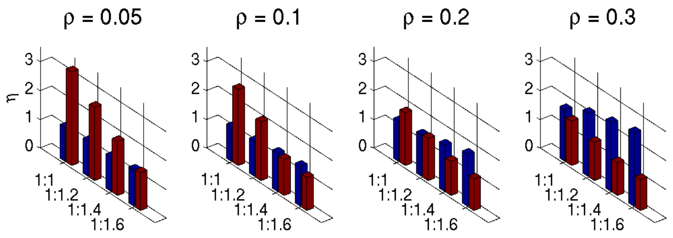
We have run and compared our method with two popular assignment algorithms: Hungarian algorithm and a greedy algorithm. The Hungarian algorithm is a deterministic centralized algorithm that always yields the global optimum, with state-of-art time complexity $O(n^3)$, hence we use it as a gold standard for comparison.



(a)



(b)



(c)

Figure 5: (a) $\eta_{p:h}$ vs ρ and $\#robots : \#tasks$; (b)$\eta_{g:h}$ vs ρ and $\#robots : \#tasks$; (c) Comparisons of assignment qualities, blue bars represent $\eta_{p:h}$ and maroon bars denote $\eta_{g:h}$ (horizontal-axis:$robots\#$: $tasks\#$; vertical-axis: η).

In this work, entirely decentralized variants of the greedy algorithm[*] operate in which robots independently evaluate the available tasks and select the maximal utility one for its assignment. At first, each robot greedily chooses the best available task for itself. If a task is simultaneously selected by more than one robot, one-round of communication among these robots are carried out to decide the winner for this task, and this task is marked unavailable for other robots. The distributed greedy algorithm is included because it provides insight into the level of quality that several practitioners have been willing to sacrifice (see Gerkey and Matarić [2004] for a detailed discussion).

Using the performance metric η as defined previously, we tested $\eta_{p:h}$ and $\eta_{g:h}$ (p, g, h denote our partitioning algorithm, greedy algorithm, and Hungarian algorithm, respectively) under different sparsity and matrix shapes for a large-scale multi-robot system of size $m+n \approx 300$. Both the Hungarian algorithm and greedy method operate directly on the original un-filtered matrix. Figures 5(a) and 5(b) show the statistics of $\eta_{p:h}$ and $\eta_{g:h}$ (note that $\eta_{p:h}$, $\eta_{g:h} \in [1, +\infty)$, and a value of η is close to 1 denotes good performance). Figure 5(a) shows that the partition based algorithm works well when ρ is small (the utility matrix is sparse) and the robot-to-task ratio is close to 1 (the utility matrix is square, *i.e.*, the number of robots and tasks are balanced). In contrast, Figure 5(b)

[*]More precisely, a greedy choice can be employed locally, which is distinct from the Greedy Algorithm but is the same in spirit.

shows that a greedy selection works well for the opposite conditions: greater numbers of utilities and the case of many redundant tasks. This means that to achieve certain accuracy, the greedy method requires much more task utility information for each robot than the partition based algorithm does. Moreover, in the region where our partitioning algorithm works well, the standard deviations for $\eta_{g:h}$ are much bigger than those for $\eta_{p:h}$, suggesting that the greedy algorithm's performance depends more critically on the particular values and ordering artifacts. Figure 5(c) combines $\eta_{p:h}$ and $\eta_{g:h}$ together, where the blue bars represent $\eta_{p:h}$ and maroon bars denote $\eta_{g:h}$, from which comparative performance is more directly observable in detail. However, as discussed in sparsity analysis, choosing too small a value for ρ may remove too many utilities and thereby reduce the assignment quality.

We have also tested the assignment performance under different numbers of partitions for a system size of around 300, while maintaining in a square shape. The result is shown in Figure 6. We analyse the $\eta_{p:h}$ as well as the total number of re-adjustments of all partitions (for correcting invalid random assignments), along with changing the number of partitions $K \in [2, 20]$. The results indicate better performance when K is small, and $\eta_{p:h}$ converges to some value around 1.6 as K increases. However, the number of re-adjustments increases monotonically with increasing K. Note that the turning point for K in the two plots is $K \approx 14$. After the turning point, $\eta_{p:h}$ essentially becomes a constant value but the number of re-adjustments becomes significant. The turning point fits the theoretical upper bound of K discussed in Section 5.3 well, *i.e.*, $K = 1/\rho(q) \approx n/N(q) = 150/10 = 15$. This indicates that for K greater than the upper bound, the performance is reduced by extensive inspections and re-adjustments of the uncertain assignments, which undermines the decentralization.

6.3 Online Assignment for Dynamic Utilities

Our algorithm is expected to perform well for online assignment problems with dynamic utilities. In order to simulate such dynamics, we permit tasks locations to "drift" with different velocities so that both robots and tasks keep moving and the utility matrix varies over time. Following procedures of Algorithm 1, a global leader is randomly selected to be responsible for aggregating the global utilities. Then the filtered utility matrix is partitioned and the resultant assignment table is distributed to a set of sub-leaders. The sub-leaders of each partition periodically (at frequency that is sufficient to capture the dynamics of the utility values) collect the utilities from other robots in the same partition. They use these to decide whether reassignment or repartitioning is necessary based on the online sparsity observations and which sub-leader(s) should be contacted to merge interrelated blocks. In our experiments, we purposefully manipulate the spatially nearby partitions to interact with each other to form interrelated blocks.

Figure 8 provides an example of how the online assignment works with dynamic tasks. In Figure 7(a) there are 5 partitions obtained by partitioning the

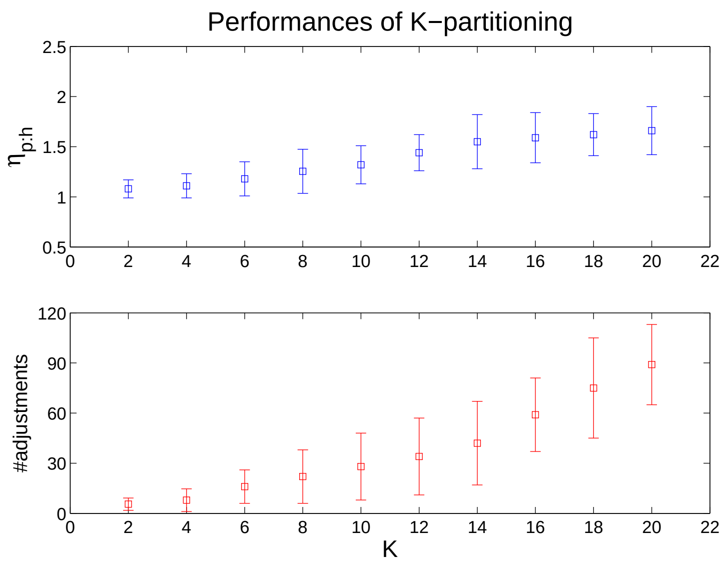
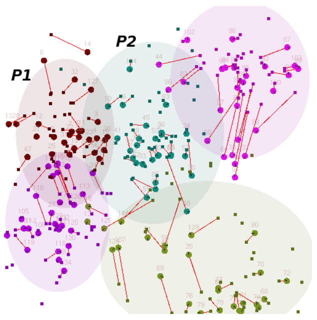


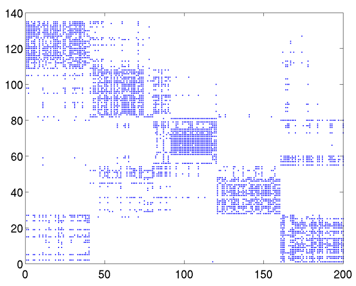
Figure 6: Partitioning-based assignment performances with regard to differing numbers of partitions.

initial sparse utility matrix; the partitioning result is easily seen in Figure 7(b). In Figure 7(c) the tasks of the first 2 partitions, P_1 and P_2, gradually diffuse into the other partitions, which correspond to the addition of non-void utility values in interrelated blocks. This is most clearly visible in Figure 7(d). This interrelationship between the partitions is detected locally and a repartitioning is prompted. The new partitions $P_1^{'}$ and $P_2^{'}$ as well as the corresponding new block matrix are shown in Figures 7(e) and 7(f), respectively.

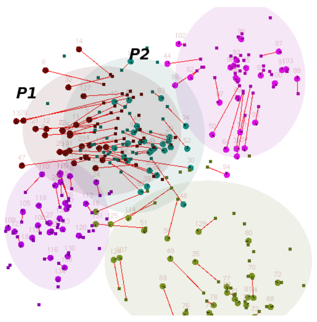
One problem specific to the online assignment is determining an appropriate threshold number N_r, which controls the frequency of restarts. Restarts become necessary after many iterations of the coalescence and repartitioning, since the initial global features gradually become stale. The impact of different N_r's are illustrated in Figure 8 from a typical online experiment with designated assignment solution quality $q = 85\%$. With regard to this solution quality, the actual sparsity $\hat{\rho}$ is observed and compared with the theoretical threshold $\rho(q)$. If the condition $\hat{\rho} < \rho(q)$ is satisfied, a repartitioning is triggered and the sub-leader's repartitioning count is incremented. The x-axis ticks show the sequence of repartitionings because we are most interested in the optima immediately after the moments of repartitioning. The y-axis denotes the actual assignment solution quality (normalized, as before, with respect to the centralized optimum). The three curves of Figure 8 are the results of $N_r = 10,\ 20,\ 30$, respectively. From the figure we can see that a very large value of N_r may cause the solution quality to deteriorate greatly. Currently, N_r is treated as an empirically determined parameter which depends significantly on the level of task dynamics. Since this



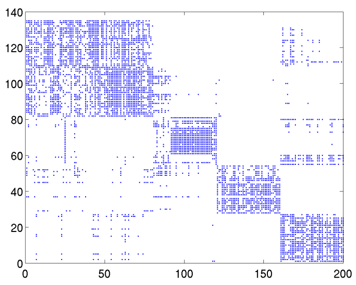
(a)



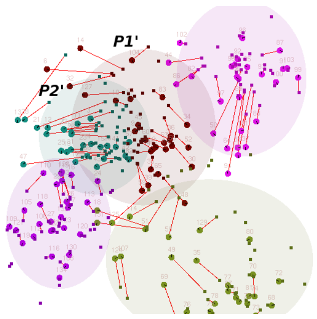
(b)



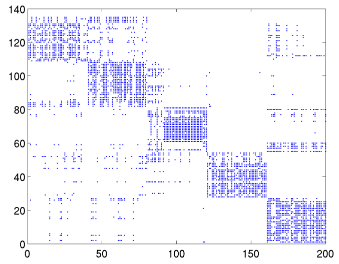
(c)



(d)



(e)



(f)

Figure 7: On-line assignment in a dynamic environment. (a)–(b) Original partitioned assignment and block matrix; (c)–(d) Partitions P_1 and P_2 are merging together, and correspondingly the interrelated blocks diffuse into each other; (e)–(f) The repartitioned $P_1^{'}$ and $P_2^{'}$ and corresponding block matrix.

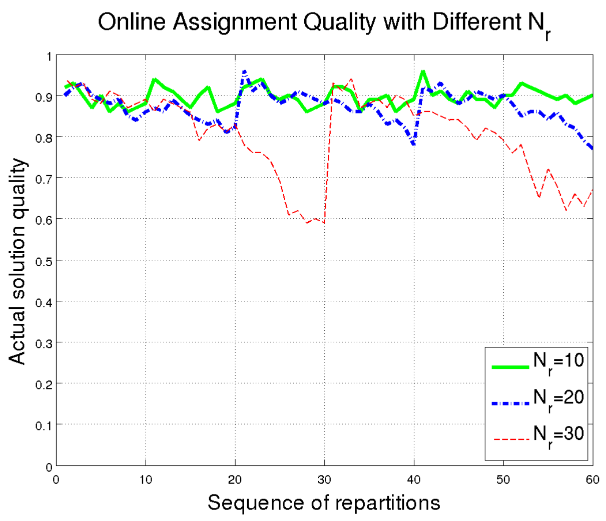


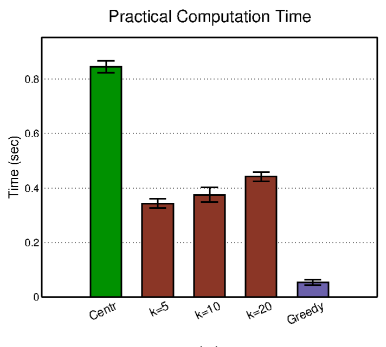
Figure 8: Online assignment qualities observed under different periodical lengths of global refreshments. A sudden increase of the solution quality indicates an operation of repartitioning.

is the online assignment problem, one good way to determine N_r is to embed it in a learning procedure so that this empirical value can be adapted online.

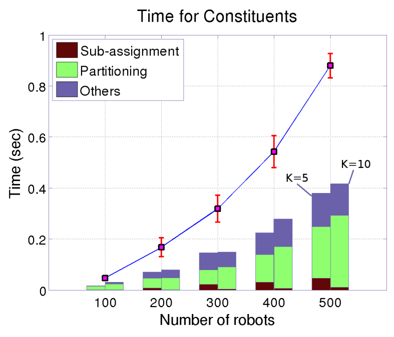
6.4 Practical Running Time

We also investigated the practical running times for the method. The initial phase involving the matrix processing and partitioning (lines 1–4 in Algorithm 1) is computed by the global leader and is the most computationally expensive step. Since it dominates the running time of an online assignment, we compare this phase with the CPU times from the centralized Hungarian algorithm (an efficient $O(n^3)$ time complexity implementation) and the distributed greedy method as we have described, respectively.

The experiments focus on the case of square matrices and employ an IBM Thinkpad laptop with 1.60GHz CPU and 2GB of memory. The `PaToH` software was used as the partitioning tool. Our experience is that `hMeTis` can be faster in practice, so the reported performance should be interpreted as reflective of the broader class of tools and can be improved with tuning. Even so, the partitioning tool is extremely fast, *e.g.,* the time to partition a 1000×1000 matrix into fewer than 10 partitions takes under 1 second. The results show that even though these partitioning computations dominate the first-phase, the overall running time is still much reduced in comparison with the optimal centralized method. Figure 9(a) presents the timing data for 500×500 matrices. The leftmost bar is the running time for the centralized method (*i.e.*, solution via the Hungarian algorithm with the state-of-art time complexity) and the rightmost



(a)



(b)

Figure 9: (a) Comparisons of practical running time among the state-of-art centralized algorithm, the greedy method, and our approach; (b) Detailed breakdown of the compute time for the different elements of the partitioning method.

bar is for the greedy method. The three bars in the center are the time for the partitioning approach, with the number of partitions labelled along the horizontal axis. (Note that in these cases, the global leader is also a member robot and is also allocated sub-assignment; time for solving this sub-assignment is included in the measurement.) We can see that the centralized algorithm is most costly, while the greedy method is the fastest, with the partitioning approach falling between them. When the number of partitions K grows, the overall time also increases slightly due to the additional cost of the partitioning operations.

Figure 9(b) presents statistics for different numbers of robots and tasks. The curve above the stacked bars shows the running time of the centralized Hungarian method. There are five groups of bars and each group gives statistics for matrices with a particular number of robots. Also, in each group, two adjacent stacked bars represent the results for $K = 5$ and $K = 10$, respectively. Every stacked bar contains three parts: recording the CPU times of the partitioning operations (middle of the stack), solving a sub-assignment (bottom of the stack), and other processing operations such as the matrix filtering (top of the stack). The figure shows that the total running time of the partitioning method is less than the fastest centralized time, even for the most expensive phase. Moreover, as K increases, the time to solve sub-assignments decreases whereas the time for partitioning increases, and that it comes to dominate the overall time. After the first-phase, the sub-assignment problems that require much less time are distributed to sub-leaders and carried out simultaneously.

7 DISCUSSION

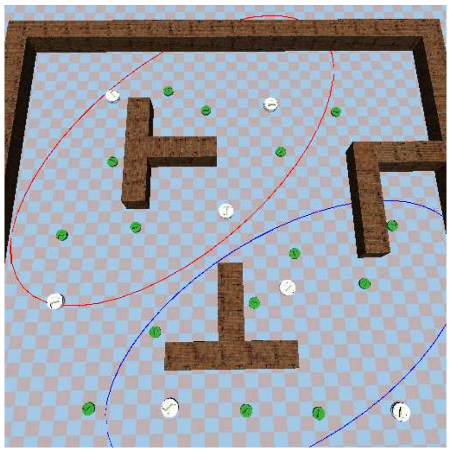
7.1 Complexity Analysis and Features of the Algorithm

If we consider the parallelism introduced via multiple decision-making agents which solve part of the assignment problem, then the partitioning algorithm improves the time complexity over the centralized solution. First, the partitioning of the initial global assignment requires $O(n^3)^\dagger$, where n denotes the number of tasks. Then, during the on-line assignment, the time complexity for each sub-leading agent to collect and filter the utility matrix as well as to check the interrelated blocks is $O(K \cdot (\frac{n}{K})^2)$; to solve the sub-assignment problem, the Hungarian algorithm costs $O((\frac{n}{K})^3)$; there may be repartitioning of the interrelated blocks, which requires $O((\frac{l \cdot n}{K})^3)$, where l is the number of interrelated diagonal blocks and $(\frac{l \cdot n}{K})$ actually denotes the column size of interrelated matrix as illustrated in Figure 2(b)). Since l usually is much smaller than K for a large system (occurring only among the neighboring partitions), and it executes less frequently (occurring only at the moment when repartitioning is triggered), we conclude that the overall running time for the dynamic assignment is generally reduced by a factor of K^3 compared with the centralized algorithm, which always requires a running time of $O(n^3)$. Note that K need not necessarily be a constant, but can be a variable as a function of the system size, *e.g.*, $K = \sqrt{n}$.

In addition, the overall communication and computation work is also greatly reduced by eliminating transmission of trivial utilities, *e.g.,* for a system involving more than 100 robots, the sparsity parameter ρ can be less than 0.1, meaning that we only communicate and compute fewer than 10% of all utilities in our assignment. Apart from the first phase, in our problem domain communication is spatially localized to other robots within the same partition. Another benefit of such sparsity processing is the improved time complexity of the centralized assignment algorithm that happens during the first phase. Since the algorithm is fed with a sparse utility matrix, the actual time complexity can be less than the given complexity of $O(n^3)$ which is produced by analyzing a complete matrix [Kuhn, 1955; Bertsekas, 1990] — *e.g.*, it may be reduced by an order if each row has small fixed number of non-zero utilities as in our case.

The introduction of multiple decision making agents also addresses the drawback of limited robustness in traditional centralized algorithms. The failure of a single sub-leading agent will not halt the whole assignment task, instead the impact of the failure remains local to a given partition. Naturally it is also possible for the failure to be automatically addressed along with the repartitioning of merged partitions.

†We assume the multi-level partitioning algorithm costs $O(n^3)$ for an $n \times n$ matrix, although it has been empirically demonstrated to be much faster [Karypis and Kumar, 1998] than the spectral partitioning method, which has a running time complexity of $O(n^3)$ dominated by computing the eigenvectors.


(a)

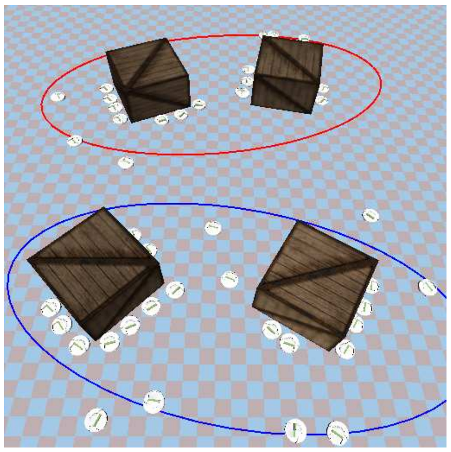

(b)

Figure 10: (a) Surveillance/monitoring a set of moving objects (#robots < #tasks); (b) Box-pushing by a team of robots (#robots > #tasks).

7.2 Applications to Robotics

The purpose of our simple simulation scenario presented in the experiments section is to clearly demonstrate the operating principles of the partitioning and distribution strategy in multi-robot task assignment problems. The approach may certainly be applied to a variety of different task allocation problems so long as the performance of individual tasks can be represented with a utility estimate and a global utility matrix can be constructed. Once the assignment problem is (recursively) partitioned into the appropriate base level of sub-assignments, different task allocation strategies for solving these sub-problems can be employed. The numbers of robots and tasks may be unbalanced in each partition. If a traditional matching method (*e.g.*, the Hungarian algorithm) is used, each robot will execute one task at one time. On completion of the assigned task, the robot removes it from the available choices and considers other tasks to perform. This is repeated until all tasks are finished. Other non-matching strategies with one-to-many or many-to-one mappings (*e.g.*, combinatorial auctions) can also be employed to solve the sub-assignments depending on specific scenarios.

The following are two examples of concrete applications that can benefit from partitioning and distributing of task allocation problems:

(1) The surveillance and monitoring of a set of moving objects by multiple mobile robots: Typically these scenarios have fewer robots than tasks and each robot need be simultaneously assigned to more than one object to monitor. Figure 10(a) illustrates a localized split of the problem so that sub-assignments that can be further partitioned or directly solved with any appropriate strategies.

(2) The allocation of a team of robots to push a group of boxes (or other large tasks requiring collaboration of multiple robots): these are representative of scenarios where the number of robots may exceed the number of tasks. An example showing a partitioned assignment is shown in Figure 10(b). In these cases, the merging and repartitioning of the sub-assignments can also be triggered by certain interruptions, *e.g.*, detection that a box is stuck on a slope or is no longer moving forward. To employ the presented approach on these particular applications is an area of future work.

Finally, our experience suggests that the method works especially well when the changes of utilities are smooth and comparatively slow; problems with extremely irregular utility changes (*e.g.*, following the step function or pulse function,*etc.*) can be improved by employing a mechanism to trigger local repartitioning using cues other than shifts in utility density.

8 CONCLUSION

In this paper, we propose a method with attractive accuracy, speed and robustness for large-scale online assignment problems. The approach employs a top-down approach which permits the problem to be distributed and information to be localized wherever possible. We demonstrated the effectiveness of the proposed algorithm and showed the efficiency with simulation experiments. The algorithm's performance is most promising for large problems when task dynamics require reallocation and for which task locality (in space and time) and sparsity can be exploited.

The algorithm described in this paper makes use of structure within the utility matrix rather than models of specific task properties. Regularity that exists due to the specific tasks being performed results in a structure for the team. Since this multi-level organization of the system is derived automatically, it can be adjusted (and re-generated) to reflect changes to tasks themselves. We believe this approach to be an important middle ground between entirely domain-independent planning mechanisms and highly specialized task-specific models.

Acknowledgments

The authors thank G. Karypis, Ü. Çatalyürek and B. Hendrickson for providing useful tools and/or suggestions.

References

[Arafeh *et al.*, 2008] B. R. Arafeh, K. Day, and A. Touzene. A Multilevel Partitioning Approach for Efficient Tasks Allocation in Heterogeneous Distributed Systems. *Journal of Systems Architecture – Embedded Systems Design*, 54(5):530–548, 2008.

[Aykanat *et al.*, 2002] Cevdet Aykanat, Ali Pinar, and Ümit V. Çatalyürek. Permuting Sparse Rectangular Matrices into Block-Diagonal Form. *SIAM Journal on Scientific Computing*, 25:1860–1879, 2002.

[Berhault *et al.*, 2003] Marc Berhault, He Huang, Pinar Keskinocak, Sven Koenig, Wedad Elmaghraby, Paul Griffin, and Anton J. Kleywegt. Robot Exploration with Combinatorial Auctions. In *Proc. of IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems (IROS'03)*, pages 1957–1962, Las Vegas, NV, U.S.A., October 2003.

[Bertsekas and Castanon, 1991] D. P. Bertsekas and D. A. Castanon. Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm. *Parallel Computing*, 17:707–732, 1991.

[Bertsekas, 1990] D. P. Bertsekas. The Auction Algorithm for Assignment and Other Network Flow Problems: A Tutorial. *Interfaces*, 1990.

[Cao *et al.*, 1997] Y. Uny Cao, Alex S. Fukunaga, and A. B. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4:226–234, 1997.

[Choi *et al.*, 2009] Han-Lim Choi, L. Brunet, and J.P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*, 25(4):912 –926, aug. 2009.

[Dhillon, 2001] Inderjit S. Dhillon. Co-clustering documents and words using Bipartite Spectral Graph Partitioning. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, pages 269–274, San Francisco, CA, 2001.

[Dias *et al.*, 2006] M. Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE — Special Issue on Multi-robot Systems*, 94(7):1257–1270, July 2006.

[Fiduccia and Mattheyse, 1982] C. Fiduccia and R. Mattheyse. A Linear Time Heuristic for Improving Network Partitions. *Proc. 19th ACM/IEEE Design Automation Conference*, 49:175–181, 1982.

[Garey *et al.*, 1974] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some Simplified NP-complete Problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.

[Gerkey and Matarić, 2004] Brian Gerkey and Maja J Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-robot Systems. *International Journal of Robotics Research*, 23(9):939–954, September 2004.

[Gilbert *et al.*, 1995] John R. Gilbert, Gary L. Miller, and Shang-Hua Teng. Geometric Mesh Partitioning: Implementation and Experiments. In *Proc. International Parallel Processing Symposium*, pages 418–427, 1995.

[Goldberg *et al.*, 2003] Dani Goldberg, Vincent A. Cicirello, M. Bernardine Dias, Reid G. Simmons, Stephen F. Smith, and Anthony Stentz. Task Allocation Using a Distributed Market-based Planning Mechanism. In *Proceedings of the International Joint Conference on Autonomous Agents & Multiagent Systems, (AAMAS)*, pages 996–997, Melbourne, Australia, 2003.

[Hendrickson and Leland, 1993] Bruce Hendrickson and Robert Leland. A Multilevel Algorithm for Partitioning Graphs. *Technical Report SAND93-1301*, 1993.

[Hendrickson *et al.*, 1998] Bruce Hendrickson, Tamara, and G. Kolda. Partitioning Rectangular And Structurally Nonsymmetric Sparse Matrices For Parallel Processing. *SIAM J. Sci. Comput*, 21:2048–2072, 1998.

[Ji *et al.*, 2006] M. Ji, S. Azuma, and M. Egerstedt. Role-Assignment in Multi-Agent Coordination. *International Journal of Assistive Robotics and Mechatronics*, 7(1):32–40, March 2006.

[Kalra and Martinoli, 2006] Nidhi Kalra and Alcherio Martinoli. A Comparative Study of Market-Based and Threshold-Based Task Allocation. In *Proc. of the International Symposium on Distributed Autonomous Robotic Systems*, Minneapolis, MM, 2006.

[Karypis and Kumar, 1998] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998.

[Kernighan and Lin, 1970] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphics. *The Bell System Technical Journal*, 49:291–307, 1970.

[Kloder and Hutchinson, 2006] S. Kloder and S. Hutchinson. Path Planning for Permutation-Invariant Multirobot Formations. *IEEE Transactions on Robotics*, 22(4):650–665, August 2006.

[Kolda, 1998] Tamara G. Kolda. Partitioning Sparse Rectangular Matrices for Parallel Processing. In *LNCS*, pages 68–79, 1998.

[Kuhn, 1955] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistic Quarterly*, 2:8397, 1955.

[Lerman *et al.*, 2006] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research*, 25:225–242, 2006.

[Liu and Shell, 2011] Lantao Liu and Dylan Shell. Assessing Optimal Assignment under Uncertainty: An Interval-based Algorithm. *The International Journal of Robotics Research*, 30(7):936–953, 2011.

[Melo and Veloso, 2011] Francisco S. Melo and Manuela Veloso. Decentralized MDPs with Sparse Interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

[Michael *et al.*, 2008] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas. Distributed Multi-Robot Task Assignment and Formation Control. In *IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

[Miller *et al.*, 1993] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic Mesh Partitioning. In Alan George et. al, editor, *Graphs Theory and Sparse Matrix Computation*, pages 57–84. Springer-Verlag, 1993.

[Papa and Markov, 2007] David A. Papa and Igor L. Markov. Hypergraph Partitioning and Clustering. In *Approximation Algorithms and Metaheuristics*, 2007.

[Pothen *et al.*, 1990] Alex Pothen, H.D. Simmon, and K.-P.Liou. Partitioning Sparse Matrices with Eigenvectors of Graphs . *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[Russell and Norvig, 2009] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc., Upper Saddle River, NJ, U.S.A., third edition, 2009.

[Simmons *et al.*, 2002] Reid Simmons, Trey Smith, M. Bernardine Dias, Dani Goldberg, David Hershberger, Anthony Stentz, and Robert Zlot. A layered architecture for coordination of mobile robots. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, 2002.

[Smith and Bullo, 2009] S. L. Smith and F. Bullo. Monotonic Target Assignment for Robotic Networks. *IEEE Transactions on Automatic Control*, 54(9):2042–2057, September 2009.

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

[Toroslu and Üçoluk, 2007] Ismail H. Toroslu and Göktürk Üçoluk. Incremental Assignment Problem. *Information Sciences*, March 2007.

[Ümit V. Çatalyürek and Aykanat, 1999] Ümit V. Çatalyürek and Cevdet Aykanat. Hypergraph-Partitioning Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Trans. on Parallel and Dist. Computing*, 10:673–693, 1999.

[Zavlanos and Pappas, 2008] M. M. Zavlanos and G. J. Pappas. Dynamic Assignment in Distributed Motion Planning with Local Coordination. *IEEE Transactions on Robotics*, 24(1):232–242, February 2008.

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

[Zlot and Stentz, 2003] Robert Zlot and Anthony Stentz. Market-based Multirobot Coordination Using Task Abstraction. In *The 4th International Conference on Field and Service Robotics*, 2003.