Multi-Level Partitioning and Distribution of the Assignment Problem for Large-Scale Multi-Robot Task Allocation

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

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

Abstract— **A team of robots can handle failures and dynamic tasks by repeatedly assigning functioning robots to tasks. This paper introduces an algorithm that scales to large numbers of robots and tasks by exploiting both task locality and sparsity. The algorithm mixes both centralized and decentralized approaches at different scales to produce a fast, robust method that is accurate and scalable, and reduces both the global communication and unnecessary repeated computation. We depart from optimization and bipartite matching formulations of the problem, observing instead that an assignment can be computed through coarsening and partitioning operations on the utility matrix. First, a coarse assignment is calculated by evaluating the global utility information and partitioning it into clusters in a problem-domain independent way. Next, the assignment solutions in each partition are refined (either recursively, or via an existing algorithm). This multilevel framework allows the repeated reassignment to execute among interrelated partitions. The results suggest that only a minor sacrifice in solution quality is required for gains in efficiency. The proposed algorithm is validated using extensive simulation experiments and the results show advantages over the traditional optimal assignment algorithms.**

I. 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 on-line 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 algorithm we describe achieves its efficiency through parallelization of large allocation calculations, and communication costs are lessened by confining messages to subsets of the team.

The algorithm we introduce is based on the observation that existing techniques for distributing jobs across multiple processors can not only be applied to distribute the allocation problem, but, in fact, recursive application of these techniques may actually be used to compute an assignment directly. The approach computes the assignment solution in two stages: the first stage is a coarse assignment where strong "connected" robot-task pairs are clustered together to produce an abstracted problem of reduced size; the second stage refines the assignment solution inside each clustered partition, *i.e.,* more accurate assignment solutions are computed inside individual partitions in a distributed fashion. By comparing with centralized and decentralized algorithms, we show that our divide-and-conquer strategy possesses several advantages, and the new formulation as a partitioning problem has potential for further improvement via future research.
The contributions of this work include:–

- Identification of a new hypergraph (and related matrix) partitioning formulation for the assignment problem which enables 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 combined and mixed up to any level of recursion, to large scale problems.
- Evaluation and exploitation sparseness properties of the utility matrix.
- A solution to what we term the "dynamic" optimal assignment where changes in subsets of the utility matrix are only propagated up to a level necessary to adjust the assignment.

II. RELATED WORK

As described in [1], task-allocation mechanisms are a form of deliberative coordination, several of which were developed in conjunction with, and as foundational elements of, multi-robot software architectures. The focus of this paper is on the most widely studied (and most widely applied) variant of this problem, which [1] terms Single-Robot-Tasks, Single-Task-Robots, involving an Instantaneous Assignment (ST-SR-IA).

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 Hungarian algorithm [2] and linear programming-based approaches fall into this family, as do some auction protocols involving an auctioneer, several examples are analyzed in [1] and [3]. *Decentralized/Distributed* 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.*, [3, 4, 5]). Important prior work has looked at parallel implementations of known centralized algorithms (*e.g.*,,[6, 7]) and under a variety of differing communication constraints (*e.g.*,, [8]). The present work attempts to combine the merits of centralized and decentralized methods: it employs global information infrequently, and even then uses that information to distribute the work in performing the task allocation.

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 [9], fine-grained robot resource sharing [10], incremental assignment [11], and sensitivity analysis [12]. The similar underlying combinatorial task-assignment problems have also been tackled from a hybrid systems perspective (*e.g.*, role assignment with preemption [13], and potential-field hybrid controllers with relaxed mutual exclusion constraints [14]). An important recent result is that of [15], 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.

III. PROBLEM DESCRIPTION

A. Sparse Matrix Model

In the envisioned scenario information is provided in the form of a *utility matrix*: entry u_{ij} represents the expected utility (or reward, or benefit) of having robot i perform task j. Since matrix transposition does not fundamentally change the problem, without loss of generality we will consider utility matrices of size $m \times n$ ($m \leq n$). We denote the ith row by r_i, which represents all utility estimates involving the ith robot. By c_j we denote the jth column, *viz.*, all estimates for performance of the jth task. An *assignment* is an association of each robot to exactly one task so that no two robots are associated with the same task. The linear 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.

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 by introducing void entries and producing a so-called *sparse* matrix representation. Such matrices can enable considerable computational and communication savings, *e.g.*, 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.* [4]), then the sparsity of the utility matrix follows from the relative positions of robots and tasks. As demonstrated by the results in Section VI-A, 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.

B. Graph Models

Besides the utility matrix representation, two graph representations (one conventional and one novel) can describe the quality of agent-task assignment pairs:

Bipartite Graph: A bipartite graph $G_B(V, E)$ with a vertex set V and an edge set E has two vertex classes X and Y, such that $V = X \cup Y$ and $X \cap Y = \varnothing$. An edge $e_{ij} \in E$ connects a vertex pair $v_i \in X$ and $v_j \in Y$. In the popular assignment problem interpretation, vertices X denote robots while Y denote tasks.

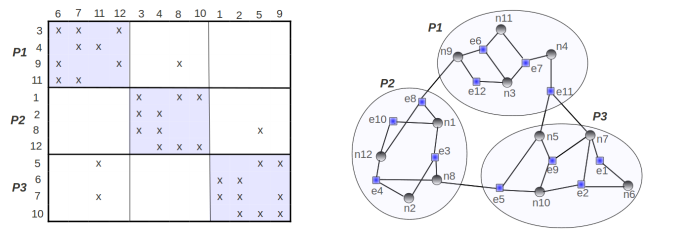
Hypergraph: A hypergraph $\mathcal{H}$ is denoted by $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes (the counterpart of vertices), and $\mathcal{E}$ is a set of non-empty subsets of $\mathcal{V}$ called hyperedges. A hyperedge generalizes the notion of an edge in standard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

For both bipartite graphs and hypergraphs the weights can be added to the edges to specify utilities, and this is done in a straightforward way. For assignment partitioning, a hypergraph is superior to a bipartite graph representation, because we can interpret the nodes of the hypergraph as robots and the hyperedges as tasks, such that the hypergraph can be regarded as a multi-robot network, as illustrated in Fig. 1.

Graphs or matrices are equivalent 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) 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. The partitioning approach we introduce is most naturally viewed in this hypergraph form.

IV. BACKGROUND: PARTITIONING

Graph partitioning is a general computational problem concerned with grouping vertices or nodes together so as to minimize the (weighted) cut-size of edges between these groups. Many research areas have specific graphs which benefit from partitioning, most relevant to this work is research on high performance computing [16, 17, 18]. Graph partitioning is known



(a) (b)

Fig. 1. (a) Partitioned sparse matrix (each "×" 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).

to be NP-complete [19], and consequently many heuristics have been proposed. *Spectral partitioning methods* and their variants (*e.g.* [20, 21, 22] have been used to partition a wide variety of graphs and specialized strategies exist in order to induce the particular properties (*e.g.*, producing balanced partitions). The Kernighan-Lin/Fiduccia-Mattheyses heuristic [23, 24] performs well in refining partitions and converges quickly when a good initial partitioning can be provided. Geometric partitioning methods (*e.g.,* [25, 26]) 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 [27, 22] combines different heuristics in its three principle phases: *coarsening*, *initial partitioning* and *uncoarsening*.

Graph partitioning results in a pattern in the corresponding matrix representation. An alternative approach is to directly partition the matrix itself via re-ordering operations: non-void entries of the sparse matrix are gathered into non-overlapping blocks on the diagonal by performing row/column swaps. This has been extensively researched for high performance computing applications [16, 17, 28] 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 [29].

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 diagonal blocks of the partitioned utility matrix represent sub-problems which can be distributed with minimal overhead. We distribute our assignment problem by partitioning the pre-processed utility matrix, which is actually accomplished by partitioning the equivalent hypergraph representation. Within the hypergraph perspective, the partitioning of a sparse matrix produces clustered sets of strongly connected nodes and hyperedges. A partitioning example showing the equivalence of the utility matrix and hypergraph representations is shown in Fig. 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).

V. 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.*, [2]) for the multi-robot task allocation. The problem is usually formalized as follows:

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

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: block-diagonalized utility matrices 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 is 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.

A. Matrix Sparsity Control

Our implementation employs a pre-processing step in order to control the degree of sparsity in the utility matrix: a 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 re-ordering.

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 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 VI) 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/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.* [15] for such a treatment).

B. Matrix Partitioning and Distribution

The matrix is partitioned by permuting rows and columns. Like parallel computing applications, balanced partitions are preferred so that the workload is equalized. The resultant K-partitioned matrix forms blocks containing approximately $\frac{m}{K}$ rows and $\frac{n}{K}$ columns and the non-void blocks finally lie on the diagonal after simple re-ordering. Let U and U_k $(k \in [1, K])$ denote the sparse utility matrix and the partitioned diagonal blocks, then ideally we have $U = diag(U_1, U_2, \cdots, U_K)$ where U_k has dimension $m^{(k)} \times n^{(k)}$, and $\sum_{k=1}^{K} m^{(k)} = m$, $\sum_{k=1}^{K} n^{(k)} = n$. We have

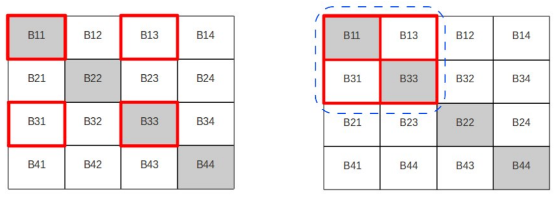
$$\sum_{j^{(k)}=1}^{n^{(k)}} x_{i^{(k)}j^{(k)}} = \sum_{j=1}^{n} x_{i^{(k)}j} = 1 \quad \forall k,\ i^{(k)} = 1, ..., m^{(k)},$$
$$\sum_{i^{(k)}=1}^{m^{(k)}} x_{i^{(k)}j^{(k)}} = \sum_{i=1}^{m} x_{ij^{(k)}} \leq 1 \quad \forall k,\ j^{(k)} = 1, ..., n^{(k)},$$

where $x_{i^{(k)}j^{(k)}}$, $x_{i^{(k)}j}$ and $x_{ij^{(k)}}$ are the binary variables defined analogously to x_{ij} but relevant to diagonal block U_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*, with each belt having $K-1$ void blocks and a single non-void diagonal block, as shown in Fig. 1(a). In each partition, we randomly pick an agent as the *sub-leading* agent, *i.e.,* there are K sub-leading agents in total. The initial assignment table is distributed to the K sub-leading agents, and the sub-leading agents are responsible for solving the independent assignment problems in their respective partitions.

C. Dynamic Assignment

The significant advantage of the initial partitioning lies in its use for solving the assignment problem when task dynamics make frequent reassignments necessary. Once an initial partitioning has been constructed for the assignment problem,



(a) (b)

Fig. 2. (a) Assignment table with interrelated blocks; (b) After row and column permutations, interrelated blocks are "clustered" into an embedded interrelated matrix which can be repartitioned again.

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 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 should be currently assigned to robots in other partitions. A repartitioning is necessary, but it need only repeated among those interacting partitions.

To assess whether repartitioning is necessary, the sub-leading robots monitor the utility density of associated blocks in the respective partitioned belts. A parameter ϕ is defined in order to describe the density change:

$$\phi = \frac{s' - s}{b_m \cdot b_n}, \tag{1}$$

where

$$s = \sum_{i=1}^{b_m} \sum_{j=1}^{b_n} I_{ij}. \quad I_{ij} = \begin{cases} 1 & \text{if entry } b_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

here s' and s are the previous and current number of non-void entries in a specific block b respectively, which has the row size b_m and column size b_n. We compare the density change of a diagonal block ϕ_d with a constant threshold $\hat{\tau} \in [0, 1]$. If $\phi_d > \hat{\tau}$, it means that the association of some agent-task pairs in this partition is weakening, *e.g.*, agents or tasks are becoming less suitable in the current partition and are better candidates for other partitions. Similarly, we define $\phi_{nd} < \check{\tau} \in [-1, 0]$ to describe the growing weight of a non-diagonal block. We term the blocks involved in such interaction the *interrelated* blocks, as the red outlined blocks in Fig. 2(a) show. Interrelated blocks are blocks for which: (1.) diagonal blocks that satisfy $\phi_d > \hat{\tau} \in [0, 1]$; (2.) non-diagonal blocks that satisfy $\phi_{nd} < \check{\tau} \in [-1, 0]$; (3.) complementary blocks to those from 1 and 2, such that all of them eventually form a rectangle with original diagonal blocks still in the diagonal. If we treat each block as an entry, then the interrelated blocks should finally form an inner *interrelated matrix* embedded in the utility matrix as illustrated in Fig. 2(b). This is because diagonal blocks can be re-positioned with a few symmetric row and column permutations.

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

D. Assignment Partitioning and Distribution Algorithm

Details from the previous subsections are captured in Algorithm 1. 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.

```
Data: Utility matrix $U$
Result: Assignment solution
Filter out and keep $\rho \cdot |cols|$ smallest utilities per row;
Make $K$ partitions ($K$ diagonal blocks) in $U$;
Distribute assignment table to each partition;
/*do below in distributed fashion*/;
forall the partitions do
    for every update (at fixed frequency) do
        Compute $\phi_d$ and $\phi_{nd}$'s;
        if interrelated partitioning required then
            Locate interrelated blocks;
            Communicate and repartition among interrelated
            blocks;
            Update assignment table;
        end
        else
            Implement reassignment locally using either
            Algorithm 1 recursively or by employing a
            centralized allocation mechanism.
        end
    end
end
```

Algorithm 1: Assignment Partitioning and Distribution

As emphasized in Section IV, 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` [27] and `PaToH` [17] perform the best (`hMeTis` is a little faster than `PaToH`). It is worth noting that our proposed assignment strategy itself—along with the partitioning implementation—includes aspects of the multilevel 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 multilevel partitioning strategy has been broadly used in many partitioning algorithms and much software, and the advantages of this framework are discussed by several authors [27, 29, 30].

In our implementation, we opted to have a single partitioning phase with the second stage operation that always employed a centralized allocation algorithm. This was because our implementation was primarily used for evaluation

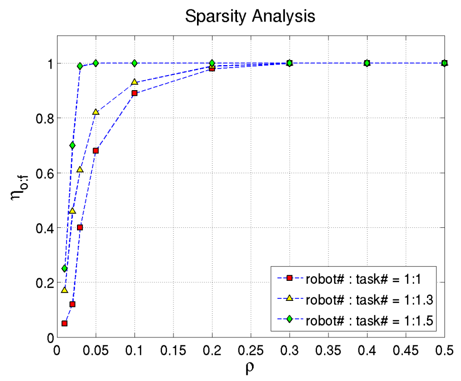


Fig. 3. Sparsity analysis showing that significant proportions of the task matrix can be discarded, with little deterioration of assignment quality.

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.

VI. EXPERIMENTS

We simulated our algorithm by considering the problem of dispatching a group of robots to a set of destination waypoints. 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, [9, 12]). 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 provided with the target locations (randomly positioned in the environment). 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.

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` [27]. 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.

A. 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 performance of x over y. In our experiments, we compute and compare $\eta_{o:f}$ (o:

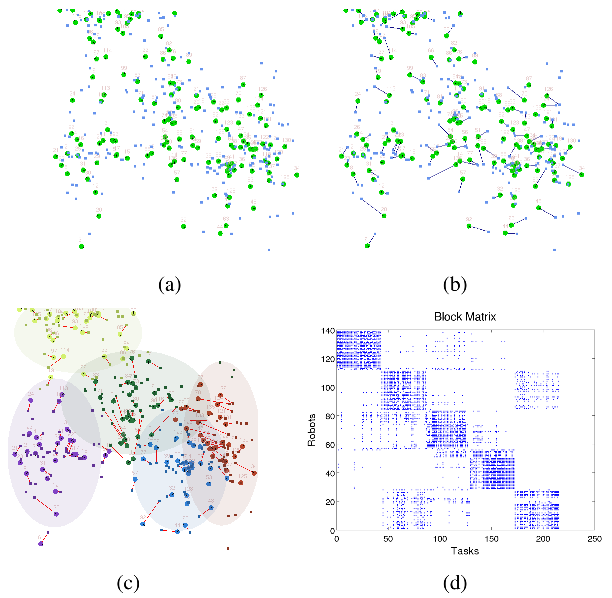


Fig. 4. (a) An assignment problem to dispatch robots to nearest task locations (circles are robots and square dots are tasks locations); (b) Centralized assignment result (lines between the robots and tasks represent the assignment); (c) Distributed assignment solution based on the partitioning results (circled areas denote partitions); (d) Block matrix of the corresponding partitions.

optimum from *original* dense matrix; f:optimum from *filtered* sparse matrix) under different conditions. Here an optimum means the distance sum of all finally assigned robot-task pairs. Our task assignment problem considers m robots and n tasks chosen so that $m + n \approx 300$. We manipulate the robot-to-task ratio in order to evaluate different matrix shapes. We also control the sparsity parameter $\rho \in [0.005, 1]$ to retain only the largest ρ times utilities in each row of the utility matrix. Fig. 3 shows the mean results of $\eta_{o:f}$ vs ρ for $\sim$20 random experiments. The curves we plotted represent typical instances: from Fig. 3 we can see that generally a good optimum can be guaranteed when $\rho \geq 0.1$ (more precisely, $\sim$15 nearest tasks for each robot contribute most to the optima). The result also shows that the more slender the matrix (with greater tasks than robots), the smaller the ρ that is permissible.

B. Assignment Partitioning for the Static Case

We have outlined in Section V that the partitioning of assignment problem can be transformed to the sparse matrix partitioning problem, therefore the first level coarse assignment is carried out by the leading robot, by partitioning the filtered utility matrix. More specifically, the sparse matrix is converted into the format of hypergraph, and fed to `hMeTiS` which 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 Fig. 4. In Fig. 4(c), there are 5 partitions obtained from the 5 diagonal blocks in the partitioned matrix in Fig. 4(d).

The robots and tasks in the same partitions/blocks are considered as new assignment problems independent of the others. The second level assignment is executed inside each partition/block in a distributed manner. In our work, we use the Hungarian algorithm [2] to solve the second level

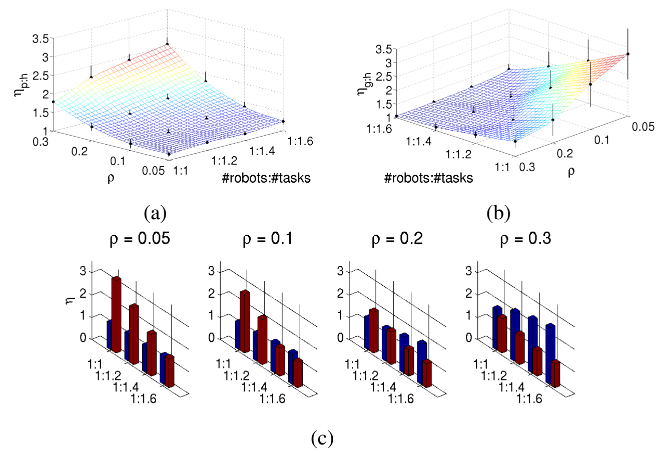


Fig. 5. (a) $\eta_{p:h}$ vs ρ and $\#robots : \#tasks$; (b)$\eta_{g:h}$ vs ρ and $\#robots : \#tasks$; (c) Comparisons of assignment qualities, blue bars represent $\eta_{p:h}$ and maroon bars denote $\eta_{g:h}$ (horizontal-axis:$robots$#: $tasks$#; vertical-axis: η).

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 equally have zero-utility. The problem lies in that the assigned entry is not the smallest in reality (the real utility before being removed), and could be sub-optimal. This 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, and if such assignments are found to be invalid, we re-adjust and thereby improve these results by greedily reassigning to the nearest available non-voided tasks.

We have run and compared our method with two other popular assignment algorithms—Hungarian algorithm and a greedy algorithm. The Hungarian algorithm is a deterministic centralized algorithm that always yields the global optimum; we use it as a gold standard for comparison. The greedy algorithm is included because it provides insight into the sacrifice of quality that several practitioners have been willing to sacrifice (see [1] for detailed discussion). Entirely decentralized variants of the greedy algorithm*exist in which robots independently evaluate the available tasks and occupies the optimal one as the assignment.

We tested $\eta_{p:h}$ and $\eta_{g:h}$ (p, g, h denote our partitioned multilevel algorithm, greedy algorithm, and Hungarian algorithm, respectively) under different sparsity and robot-to-task ratios. Fig. 5(a) and 5(b) show the statistics of $\eta_{p:h}$ and $\eta_{g:h}$ ($\eta_{p:h}$, $\eta_{g:h} \in [1, +\infty)$. For all these data, a value of η is close to 1 denotes good performance. Fig. 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, Fig. 5(b) shows that a greedy selection works well for the opposite conditions: greater numbers of utilities and the case of many redundant tasks. Moreover, in the region where our partitioning algorithm works well, the standard deviations for $\eta_{g:h}$ are much bigger than those

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

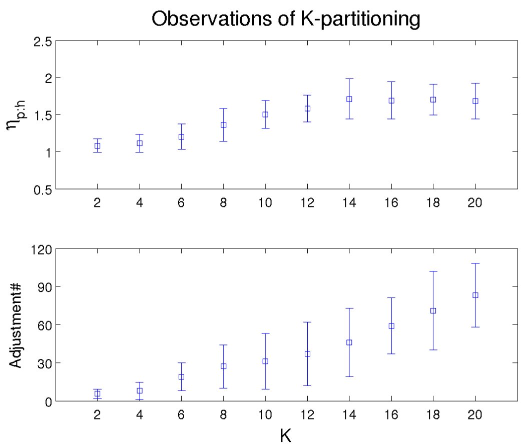


Fig. 6. Observations with regard to differing numbers of partitions.

for $\eta_{p:h}$, suggesting that the greedy algorithm's performance depends more critically on the particular values and ordering artifacts. Fig. 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. This figure also emphasizes the effect that ρ has on the methods: choosing too small a value removes too many utilities reducing the assignment quality. In our experiments the results are accurate when $\rho \approx 0.1$ (more precisely, $\sim$15 neighboring tasks).

We have also tested the assignment performance under different number of partitions, as illustrated in Fig. 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, suggesting greater correction is required.

C. On-line Assignment in Dynamic Environment

Our algorithm is expected to be superior for handling the on-line assignment problems in dynamic environments. To simulate the dynamic change of utilities, we permit tasks locations to "drift" with different velocities. The sub-leading robots of each partition periodically collect the utilities from other member robots in the same partition, and decide whether reassignment or repartitioning is necessary. In our experiments, we set $\hat{\tau} = .4$ and $\check{\tau} = -.2$ to capture the interrelated blocks. If interrelated blocks are detected, their sub-leading robots communicate with each other and do the repartitioning only among these interrelated blocks; otherwise the sub-leading robots simply run the Hungarian algorithm to solve the assignment inside their independent partitions.

Fig. 7 provides an example of how the on-line assignment works with dynamic tasks. In Fig. 7(a) there are 5 partitions obtained from partitioning the initial sparse utility matrix, which has a partitioning result shown in Fig. 7(b). In Fig 7(c) the tasks of the first 2 partitions, P_1 and P_2, gradually diffuse into the other partitions, which corresponds to the addition of non-void utility values in interrelated blocks. This is most clearly visible in Fig. 7(d). This interrelationship between the

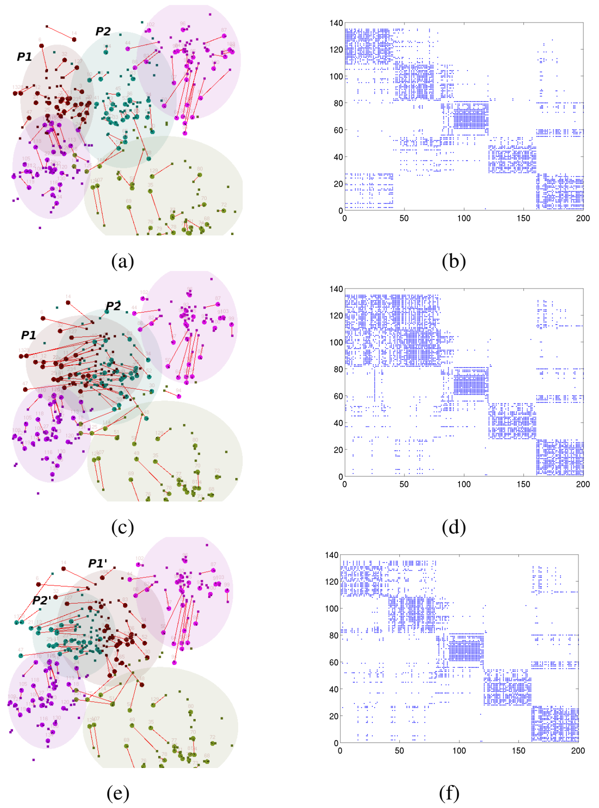


Fig. 7. On-line assignment in a dynamic environment. (a)–(b) Original partitioned assignment and block matrix; (c)–(d) Partitions P_1 and P_2 are merging together, and correspondingly the interrelated blocks diffuse into each other; (e)–(f) The repartitioned $P_1^{'}$ and $P_2^{'}$ and corresponding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

partitions is detected locally and repartitioning prompted. The new partitions $P_1^{'}$ and $P_2^{'}$ as well as the corresponding new block matrix are shown in Fig. 7(e) and 7(f), respectively.

We have also compared the performance of repartitioning among only the interrelated blocks versus the all blocks. The results show that the optimum repartitioning among interrelated blocks is only 5%–10% worse than that of among all blocks. However, the interrelated block partitioning algorithm significantly reduces the communication and computation complexity since the number of interrelated blocks are usually fewer than the total number of all blocks.

VII. DISCUSSION

Our algorithm improves the time complexity over the centralized solution in that it introduces parallelism with multiple decision-making agents which solve part of the assignment problem. First, the partitioning on 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 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 $O((\frac{d \cdot n}{K})^3)$, where d is the number of interrelated diagonal blocks and $(\frac{d \cdot n}{K})$ actually denotes the column size of interrelated matrix as illustrated in Fig. 2(b)). Since d usually is much smaller than K, and executes less frequently, we conclude that the overall running time of dynamic assignment is generally reduced by a factor of K^3 compared to the centralized algorithm which always requires a running time of $O(n^3)$.

In addition, the overall communication and computation work is also greatly reduced by eliminating transmission of trivial utilities, *e.g.,* we only communicate and compute approximately 10% ($\rho \approx 0.1$) of all utilities in our assignment experiments. Apart from the first-phase, communication in our problem domain is spatially localized to other robots within the same partition.

Moreover, the introduction of multiple decision making agents also addresses the drawback of limited robustness in traditional centralized algorithm. Failure of a single sub-leading agent will not halt the whole assignment task. It is also possible for the failure to be automatically fixed along with the repartitioning of merged partitions.

Finally, to guarantee the accuracy of global assignment, we suggest that, a fresh global new partitioning could be implemented after some time period. 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. Such a mechanism could also be embedded in a learning procedure so that the appropriate parameters, such as ρ and K, can be dynamically adjusted for specific accuracy requirements.

VIII. CONCLUSION

In this paper, we propose a method with attractive accuracy, speed and robustness for large-scale on-line 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.

ACKNOWLEDGMENTS

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

REFERENCES

[1] B. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, Sept. 2004.

[2] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, p. 8397, 1955.

[3] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: a survey and analysis," *Proceedings of the IEEE—Special Issue on Multi-robot Systems*, vol. 94, no. 7, pp. 1257–1270, July 2006.

[4] N. Kalra and A. 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.

[5] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 650–665, Aug. 2006.

[†]We assume the multilevel partitioning algorithm costs $O(n^3)$ for an $n \times n$ matrix, although it has been empirically demonstrated to be much faster [27] than the spectral partitioning method, which has a running time complexity of $O(n^3)$ dominated by computing the eigenvectors.

[6] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems: A tutorial," *Interfaces*, 1990.

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

[8] 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*, Cancun, Mexico, Dec. 2008, pp. 1212–1217.

[9] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. J. Kleywegt, "Robot Exploration with Combinatorial Auctions," in *Proc. of IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems (IROS'03)*, Las Vegas, NV, U.S.A., Oct. 2003, pp. 1957–1962.

[10] 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)*, 2007, pp. 3351–3358.

[11] I. H. Toroslu and G. Üçoluk, "Incremental assignment problem," *Information Sciences*, March 2007.

[12] L. Liu and D. Shell, "Assessing optimal assignment under uncertainty: An interval-based algorithm," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[13] M. Ji, S. Azuma, and M. Egerstedt, "Role-Assignment in Multi-Agent Coordination," *International Journal of Assistive Robotics and Mechatronics*, vol. 7, no. 1, pp. 32–40, Mar. 2006.

[14] M. M. Zavlanos and G. J. Pappas, "Dynamic Assignment in Distributed Motion Planning with Local Coordination," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 232–242, Feb. 2008.

[15] S. L. Smith and F. Bullo, "Monotonic Target Assignment for Robotic Networks," *IEEE Transactions on Automatic Control*, vol. 54, no. 9, pp. 2042–2057, Sept. 2009.

[16] T. G. Kolda, "Partitioning sparse rectangular matrices for parallel processing," in *LNCS*, 1998, pp. 68–79.

[17] Ümit V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. on Parallel and Dist. Computing*, vol. 10, pp. 673–693, 1999.

[18] 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*, vol. 54, no. 5, pp. 530–548, 2008.

[19] 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*, 1974, pp. 47–63.

[20] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2001, pp. 269–274.

[21] A. Pothen, H. Simmon, and K.-P.Liou, "partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Anal. Appl.*, vol. 11, pp. 430–452, 1990.

[22] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Sandia National Laboratories, 1993.

[23] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphics," *The Bell System Tech. J.*, vol. 49, pp. 291–307, 1970.

[24] C. Fiduccia and R. Mattheyse, "A linear time heuristic for improving network partitions," *Proc. 19th ACM/IEEE Design Automation Conference*, vol. 49, pp. 175–181, 1982.

[25] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, "Automatic mesh partitioning," in *Graphs Theory and Sparse Matrix Computation*, A. George et. al, Ed. Springer-Verlag, 1993, pp. 57–84.

[26] J. R. Gilbert, G. L. Miller, and S.-H. Teng, "Geometric mesh partitioning: Implementation and experiments," in *Proc. International Parallel Processing Symposium*, 1995, pp. 418–427.

[27] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, pp. 359–392, 1998.

[28] C. Aykanat, A. Pinar, and Ümit V. Çatalyürek, "Permuting sparse rectangular matrices into block-diagonal form," *SIAM Journal on Scientific Computing*, vol. 25, pp. 1860–1879, 2002.

[29] B. Hendrickson, Tamara, and G. Kolda, "Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel processing," *SIAM J. Sci. Comput*, vol. 21, pp. 2048–2072, 1998.

[30] D. A. Papa and I. L. Markov, "Hypergraph partitioning and clustering," in *Approximation Algorithms and Metaheuristics*, 2007.