Decentralized Task Assignment in Camera Networks

Angelo Cenedese, Federico Cerruti, Mirko Fabbro, Chiara Masiero and Luca Schenato

Abstract— **The problem of task assignment naturally arises in multiagent multitask systems, where an optimal matching of agents and tasks is sought. This problem is a combinatorial optimization problem that can be solved in a centralized fashion, given the knowledge over all agent states and available tasks and the presence of a coordination center. Nonetheless, a decentralized solution would be beneficial in systems where communication among all agents may be limited and where the system efficiency and robustness cannot rely on the performance of a single central unit. A typical example of such a system is a videosurveillance network. In this work, we address the problem of decentralized task assignment in this context, first by formalizing the problem and presenting a general model framework, and then by proposing a decentralized solution inspired by the Stable Marriage Problem, that presents interesting performances in terms of optimality over a number of defined metrics, being supported by comparison with both centralized and decentralized algorithms implemented in numerical simulations.**

I. Introduction

A modern trend in applied research concerns the application of the distributed paradigm to manage complex system, such as for example, the case of monitoring and surveillance networks [1].

The employment of a decentralized approach in such applications appears beneficial with respect to centralized ones, because it can ease modularity and flexibility of the system, dynamic reconfiguration of the network, scalability, and inherent robustness in terms of failure of components and security against malicious breaching.

Although the approach presented in this work is of general scope, we will devote particular attention to a distributed camera network for videosurveillance, and the following problem will be in focus: given a set of agents widely distributed over the area to be monitored, subject to agent resource limitations (e.g.: exclusive access mass storage or limited Pan-Tilt-Zoom camera speed), and given a set of tasks the network has to complete, we are interested in how to assign tasks and coordinate behaviors among the agents. In this framework, the contribution of this work is twofold: **(a)** to formalize the task assignment problem for a multi-agent multi-task finite resource network and **(b)** to design a decentralized procedure to attain task assignment.

The remainder of the paper is organized as follows: in Sec. III the multiagent network definition and the problem of task assignment are stated formally, and in Sec. II a brief overview on related works is given. Then, Secs. IV-V contain the core of the work, respectively dealing with task assignment modeling and control, and Sec. VI present the evaluative simulations. Finally, in Sec. VII some conclusions are drawn.

A.Cenedese is with the Department of Engineering and Management, University of Padova, 36100 Vicenza, Italy

F.Cerruti, M.Fabbro, C.Masiero and L. Schenato are with the Department of Information Engineering, University of Padova, 35131 Padova, Italy

The research leading to these results has received funding from the European Community's Seventh Framework Programme under agreement n. FP7-ICT-223866-FeedNetBack.

II. State of the Art

Applications of the task assignment problem can be found in many different disciplines, ranging from the information technology and computer science, to business administration and project management.

To begin with, a naive approach is frequently used in these applications because of its simplicity of implementation. Such an approach can be applied to the multiagent surveillance networks simply by attaining combinations of task and agents on a first-come first-served policy: a new task instance is assigned to the first free agent that is compatible with the task. In case of assignment conflicts, an elected leader agent is deputed to solve the impasse. Obviously, this approach does not give any guarantees on the quality of assignment but it is fast and robust to node failure, and moreover the presence of a leader agent prevents deadlocks.

On a different spirit, the market-based approach relies on the idea that it is possible to obtain a match between agents and tasks by means of auctions. In its simplest incarnation, agents play the role of bidders to gain the task assignment. These methods are for instance frequently used in coordinating mobile vehicles in search-and-rescue missions [2][3][4].

Also, similar scenarios may arise in the coordinated information discovery problem. Suppose that a set of agents is exploring the environment and looking for pieces of information that are spread around: the optimal association among agents and information bits is sought in order to minimize the cost of information discovery. In this case though, the main difference stands in the dynamics of the system, since the occurrence of new information happens on a time scale that is much slower than the agent convergence to knowledge [5].

A further point of view looks at game-theoretical approaches [6]. A typical situation can be the assignment of tasks to employees. The key idea is to take into consideration the preferences of both managers and employees. The issue is formulated in order to resume a weighted multiple knapsack problem, in which tasks play the role of items, and each employee work can be viewed as a knapsack to fill.

To complete this brief and not exhaustive overview, the assignment problem can be thought as a Stable Marriage

Problem [7][8] that can be handled by means of Gale-Shapley algorithms [9]. This solution has been found particularly interesting in this work and constitutes the base for the development of the proposed strategy. It will be developed in the following sections.

III. Statement and Formalization of the Camera Network Problem

We consider n agents $\mathcal{A} = \{a_1, \ldots, a_n\}$ distributed in the environment and for simplicity interconnected by a meshed communication graph, meaning that local information is known by all agents across the network. In other terms, each agent is capable of taking autonomous decision based on the complete information over the state of the network and the environment: the system design is therefore referred to as decentralized (Fig. 1).

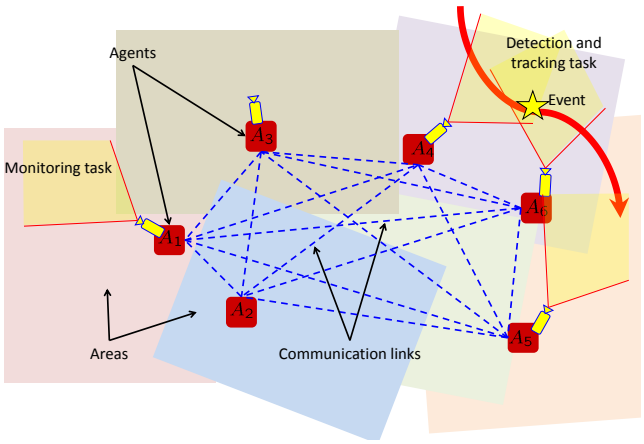


Fig. 1: Surveillance scenario. Agents are spread over the environment divided into areas, and are interconnected by a communication network. Monitoring and event detection/tracking tasks are graphically shown.

The environment is also partitioned into p areas of interest that are assumed to coincide with the field of views of at least one camera (in the following, we will reasonably assume that a minimum level of redundancy is present in the scene): in doing so, a binary coverage matrix $V \in \mathbb{R}^{n \times p}$ is defined where the non-null entries along the i-th row indicate the areas actually seen by the i-th agent, while the non-null entries along the j-th column indicate the areas that can see the j-th region. [1]

At the same time, a set of s different asynchronous task functions can be issued to the network: $\mathcal{T} = \{\tau_1(\cdot), \ldots, \tau_s(\cdot)\}$. Without loss of generality, in this work each task is characterized by type, target area or target agent, intrinsic priority $\mathrm{pr}(\tau_j) \in (0, 1]$, and drop time T_{drop}, after which the task can be considered obsolete. The instance of a task is also time-stamped with its occurrence time T_{occ}.

Finally, when a task $\tau_j(\cdot)$ is generated, one key characteristic is the list of agents that can fulfill it, $\mathcal{A}(\tau_j) \subseteq \mathcal{A}$: conversely, each agent a_i sees a list of possible task to undertake $\mathcal{T}(a_i)$.

In particular, in the following a simplified but perfectly sensible task set is considered, such as

- automatic tracking $\tau_1(\cdot)$: area related task; its request is pulsated by a camera that has detected an anomalous event in an area; the intrinsic priority of the task is medium;
- manual tracking $\tau_2(\cdot)$: agent related task; it has high priority and receives direct commands from an operator;
- patroling task $\tau_3(\cdot)$: area related task; it is a default task (zero priority) although of crucial importance: if an anomalous event takes place (i.e. an event detection occurs), the patroling agent issues an automatic tracking request to the network;
- playback streaming $\tau_4(\cdot)$: area related task; agents are used as a distributed repository of video information retrieved from a specific area; it is given low priority.

To exemplify the idea and state it clearly, the task of streaming, say $\tau_1(\cdot)$, could be issued explicitly for two different agents a_1 and a_2 or implicitly for the geographical area seen by a_1 and a_2, with exactly the same result of generating tasks $\tau_1(a_1)$ and $\tau_1(a_2)$.

Given this specific task list, the maximum number of different tasks that can be issued to the network is given by $l = 3p + n$, being p the number of possible different automatic tracking, playback, or patroling requests (one per each area), and n that of possible manual tracking requests (one per each agent).

As a final remark, it is noted that to allow for continuity in the task management and actually deal with a realistic scenario, a level of redundancy is enforced in the system, namely that for each tracking or patrolling task there are at least two cameras that are able to cover the same area (i.e. $|\mathcal{A}(\tau_1)| \geq 2$). This implies that each column of the coverage matrix V presents at least two 1's.

IV. Task Assignment Modeling

Two data structures are defined for task management: a *Waiting Task List* (WTL) and an *Active Task List* (ATL). Let define the task pool as $ATL \cup WTL$. When a task occurs, it is initially added to the WTL, and then it is either moved to the ATL if engaged in by an agent or it is dropped off if obsolete. In this respect, it is meaningful to define a dropping procedure that contrasts obsolescence: when a task exceeds a fixed lifetime, it is removed although not completed or even never undertaken, because it can be reasonably considered unprofitable and in doing so a WTL excessive growth is avoided. These two lists are thought to be either global network structures, or equivalently local agent structures (WTL(a_i) and ATL(a_i), for agent a_i, obtained from exchanging activity information within the neighborhood and from capturing the neighborhood relevant tasks issued to the network. By doing so, at every instant all agents are aware of the existing tasks that may be of interest and of their characteristics.

[1] In the context of this work the problem of optimal area coverage is not taken into account, although of relevance.

The main issue when devising a procedure to solve the task assignment problem basically resides in its time-varying nature: in fact, tasks can be generated with a rate that is faster than the average completion time, so that the assignment is always dynamic and (in principle) may never reach a steady state.

More formally: at a fixed time t, given the set of n agents and the presence of m_t task instances, to solve the task assignment problem means to find a binary array $\mathbf{x}_t \in \mathbb{R}^{nm_t}$ that maximizes some utility function $J(\mathbf{x}) : \mathbb{R}^{nm_t} \rightarrow \mathbb{R}$, with respect to the constraint $A_t\mathbf{x} \leq \mathbf{b}$.

In detail,

$$\mathbf{x}_t = [x_{1\,1}\, x_{1\,2}\, \ldots\, x_{2\,1}\, x_{2\,2}\, \ldots\, x_{n1}\, x_{n\,2}\, \ldots\, x_{n\,m_t}]^\top$$

where the variable x_{ij} assumes unitary value if the j-th task is given to the i-th agent, whereas it is equal to zero otherwise.

The binary matrix $A_t \in \mathbb{R}^{*\times nm_t}$ is related to the specific problem constraints, and shows as many rows as the number of constraints (whose values are given by the $\mathbf{b}$ vector). In general, the constraints fall into three categories, namely those related to the agents activity (e.g. one task per agent), those related to the task fulfilment (e.g. one agent per task), and those related to the feasibility of the agent-task assignment (e.g. constraints dictated by the coverage matrix). In the particular case of interest, the constraints are, for task and agent respectively:

$$\sum_{i=1}^{n} x_{ij} \leq 1 \quad \forall j = 1, \ldots, m_t, \tag{1}$$

and

$$\sum_{j=1}^{m_t} x_{ij} \leq 1 \quad \forall i = 1, \ldots, n. \tag{2}$$

For the feasibility of the coverage assignment, assuming that all agents are equally able to perform all other tasks, the constraint takes the following form:

$$\sum_{i=1}^{n} \bar{v}_{ih} x_{ih} = 0 \quad \forall h = 1, \ldots, p, \tag{3}$$

being $\bar{v}_{ih}$ the one's complement of the V-matrix entry $\bar{v}_{ih}$.

By accounting only for Equations (1) and (2), the A_t matrix is of size $(n + m_t) \times (nm_t)$ can be therefore written as

$$A_t = \left[\begin{array}{c|c|c|c} \mathbf{1} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & \ldots & \mathbf{1} \\ \hline I_{m_t} & I_{m_t} & \ldots & I_{m_t} \end{array}\right], \tag{4}$$

where $\mathbf{0}$ and $\mathbf{1}$ are m_t-dimensional row vectors of zeros and ones respectively, and I_{m_t} is clearly the $m_t \times m_t$ identity matrix. Most importantly, in this form the A_t matrix is totally unimodular (TU), i.e. it is a binary matrix whose determinant is ± 1: this condition actually yields the nice property that the integer constraints on the solution $x_{ij} \in \{0, 1\}$ can be relaxed to $x_{ij} \in [0, 1]$, therefore $\mathbf{x}$ can be defined as a continuous variable. In other words, the total unimodularity of the constraint matrix guarantees that the continuous optimal solution to the problem coincides with the discrete optimal solution in all feasible $\mathbf{x}$. Differently, the constraints defined by Eqn (3) in the A_t matrix formulation would lead to a non TU matrix, thus leading to a complex combinatorial problem. As a consequence, we propose to overcome this difficulty by implicitly taking into account the constraints of Eqn. (3) by properly defining the utility function to be maximized, thus removing them from the constraints. In particular we define the utility function as a linear combination of the variable terms x_{ij}:

$$J(\mathbf{x}, t) = \mathbf{c}(t)^\top \mathbf{x} = \sum_{i=1}^{n} \sum_{j=1}^{m_t} c_j(t) x_{ij}, \tag{5}$$

where if a particular task-agent assignment is feasible, for example w.r.t. the information given by the coverage matrix, the weight $c_j(t)$ is set to a positive value related to the priority of the task, otherwise it is set to $-\infty$.

Starting from these assumptions, some considerations are in order. A feasible solution vector $\mathbf{x}$ is chosen among $z_t = 2^{nm_t}$ different strings $\mathcal{Z}_t$: instead, the cardinality of the feasible solution set $\mathcal{X}_t \subset \mathcal{Z}_t$ (w.r.t. constraints (2)-(1)) is given by

$$|\mathcal{X}_t| = \sum_{k=0}^{\min(n,m_t)} \left(\begin{array}{c} n \\ k \end{array} \right) k! \left(\begin{array}{c} m_t \\ k \end{array} \right) \tag{6}$$

In these terms, the problem is perfectly symmetric in the agent-task duality, being k the number of allocated agents/undertaken tasks.

Also, if a default task is assumed for all agents (e.g. the patroling task), then at any instant no agent can remain unassigned in the network, and the cardinality of $\mathcal{X}_t$ is simply

$$|\mathcal{X}_t| = \min(n, m_t)! \left(\begin{array}{c} \max(n, m_t) \\ \min(n, m_t) \end{array} \right) \tag{7}$$

It is now interesting to study the behavior of this set, when the list of tasks to be fulfilled is dynamic. An effective pictorial view of the set $\mathcal{Z}_t$ is given by the vertices of the unitary hypercube in $\mathbb{R}^{nm_t}$, while the feasible solution set $\mathcal{X}_t$ is obtained by intersecting the hypercube with the constraint hyperplane (Fig. 2).

A. Dynamic Task Occurrence

It is intuitive to understand that when a new task instance occurs, the solution space dimension increases to $\mathbb{R}^{n(m_t+1)}$ and the previous solution set $\mathcal{X}_t$ remains feasible also for the new task configuration although with the new task still unassigned (Fig. 2): hence, in general, this solution may not be optimal because very different scenarios can take place. For example, it is possible that the occurrence of a new task does not imply any modification to the optimal solution (for instance, if all agents of the interested area are already performing more convenient tasks); or else, the new occurrence may cause many agents to change their tasks with better ones that are now available.

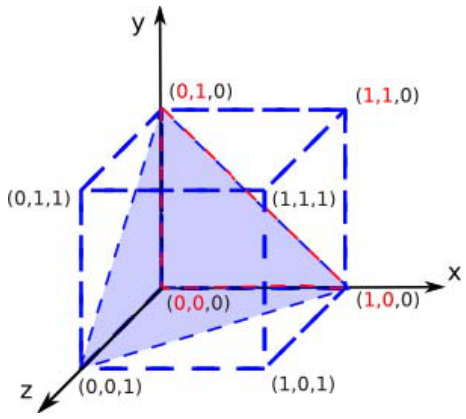


Fig. 2: Task-agent assignment. With $n = 1$ agent and $m_t = 2$ tasks, the vertices of a square represent the set $\mathcal{Z}_t$ while the intersection with the diagonal highlights the feasible set $\mathcal{X}_t$. As the number of tasks increases ($m_t = 3$), the set $\mathcal{Z}_t$ grows to the cube vertices and the feasible set $\mathcal{X}_t$ is obtained by intersection with the diagonal plane.

Conversely, when a task becomes obsolete or is completed, the dimension of $\mathcal{Z}_t$-space reduces to $(m_t - 1)n$. In the latter case, the task belongs to the WTL and thus the optimal agent-task association is not affected. In the former case, the optimal arrangement changes and two possible choices can be made for the assignment problem: either the assignment is recomputed for the new task set and the whole agent set, which can guarantee the optimality of the solution, or it is based heuristically on a greedy local policy, in the sense that the agent that has freed its resources takes on the most beneficial unassigned task. This kind of heuristics can not ensure optimality.

B. Performance Metrics

The intrinsic complexity of the studied application and the necessity of working with dynamic assignments, imply heterogeneous concurrent and often contrasting goals. On the one side, an optimal assignment is sought w.r.t. task priorities, on the other one a certain continuity in tasks execution is beneficial, avoiding situations where tasks are temporarily paused even if almost fulfilled, or repeatedly assigned to different agents before completion.

In this context, the following performance metrics have been considered, both referring to the assignment solution $\mathbf{x}$ and to the whole system design:

- optimality: we define the solution optimality $P_T(\mathbf{x}_t)$ in the time interval $[0, T]$ as

$$P_T(\mathbf{x}_t) = \frac{1}{T} \int_0^T \frac{\sum_{\tau_i \in ATL} \mathrm{pr}(\tau_i)}{\max_j \sum_n \mathrm{pr}(\tau_j)} dt \tag{8}$$

  where $\max_j \sum_n \mathrm{pr}(\tau_j)$ is the sum of the n higher intrinsic priority values in $ATL \cup WTL$, and for sake of notation we used $\tau_i = \tau_i(t)$ and $ATL = ATL(t)$; thus, $P_T(\mathbf{x}_t)$ states how close solution $\mathbf{x_t}$ is to the ideal best choice w.r.t. the feasibility of task assignment;

- idle state avoidance: from a different point of view than that of optimality, it can happen that not optimal assignments cause some agent to be in idle state, in the sense that even the possible area patroling has been taken by some sibling. We model the patroling activity as a task in order to achieve a better uniformity among tasks[2]. With regard to the utility function of Eqn. (5), it is possible to say that, if patroling tasks are enough to keep all the agents busy, each optimal solution is characterized by the fact that all agents are busy. A figure of the overall idle state of the network is given by:

$$I_T(\mathbf{x}_t) = \frac{1}{T} \int_0^T \sum_{i=1}^{n} \left(\overline{\sum_{j=1}^{m_t} x_{ij}} \right) dt \tag{9}$$

  where in reality $x_{ij} = x_{ij}(t)$ and the bar indicates the one's complement.

- assignment interruption: from a local perspective, a continuous task swapping may be detrimental for the agent, therefore we define an index that tries to capture how much each agent completes by itself the assigned tasks. For this purpose, we introduce the following index:

$$D_T(\mathbf{x}_t) = \frac{1}{T} \int_0^T \sum_{i=1}^{n} d_i(\mathbf{x}_t)\, dt \tag{10}$$

  where $d_i(\mathbf{x}_t)$ is an indicator function triggered to 1 each time a task is sent to the WTL before completion;

- average waiting time: although the permanence of high-priority tasks in the WTL strongly depends on new tasks arrival rate, it also suggests system inefficiency. Let $\mathcal{T}_T^C(\mathbf{x}_t)$ be the set of task completed in the $[0, T]$ time interval, we introduce the average waiting time:

$$W_T(\mathbf{x}_t) = \frac{1}{|\mathcal{T}_T^C(\mathbf{x}_t)|} \sum_{j=1}^{|\mathcal{T}_T^C(\mathbf{x}_t)|} T_{WTL,j}(\mathbf{x}_t), \tag{11}$$

  where $T_{WTL,j}$ is the permanence time of the j-th task in the WTL, that depends on the task assignment $\mathbf{x_t}$;

- dropping rate: being $\mathcal{T}_T^D(\mathbf{x}_t)$ be the set of task dropped and $\mathcal{T}_T(\mathbf{x}_t)$ the set of task other than patroling, in $[0, T]$, the dropping rate is defined as

$$F_T(\mathbf{x}_t) = \frac{|\mathcal{T}_T^D(\mathbf{x}_t)|}{|\mathcal{T}_T(\mathbf{x}_t)|}. \tag{12}$$

V. Task Assignment Control

A. The Gale-Shapley Algorithm

The assignment issue can be thought as a Stable Marriage Problem (SMP) formalized by Gale and Shapley [9] [7]. An instance of the classical SMP involves N men and N women, each of whom builds a preference list containing all the members of the opposite sex in a strict decreasing order. The goal is to find a set of stable marriages between the

[2]Our approach creates an initial pool whose elements are patroling tasks, one for each area that has to be monitored. A sufficient condition to assure that patroling tasks are enough to keep all agents busy is that the covering matrix $\mathbf{V}$ contains a permutation of the columns of the identity matrix I_N

men and the women. A marriage is said to be stable if there is no pair of a man and a woman who both prefer another partner to their current one. The algorithm proposed in [7] to solve this problem is the following: an unpaired man θ_1 considers the first woman ϕ on his list and removes her from it. If ϕ is not engaged, she accepts his proposal, otherwise she evaluates it: if she prefers θ_1 to her current partner θ_0, she breaks up with θ_0 (who gets unpaired) and marries θ_1, otherwise θ_1 is still unpaired because ϕ is happier with θ_0. These operations are repeated while there are unpaired men. The termination of this algorithm is assured, thanks to the elimination of one woman from one man's list during every iteration. The attained marriage is stable and optimum for men, i.e. they are paired with their highest preferred woman among the possible stable solutions.

Nevertheless, a direct application of this algorithm to video-surveillance case study problem is not possible, because of dynamics and characteristics of tasks and agents. These features lead to deal with one of the variants of SMP: the Stable Marriage Problem with Ties and Incomplete Lists (SMTI) [8]. The main differences with SMP are that lists can be incomplete, i.e. each person has a list that consists of a subset of the members of the opposite sex, and they contain ties, i.e. a man(woman) can be indifferent among two or more women(men) who will be placed in the same position of his(her) list. In this context, a marriage $\mathcal{M}$ is defined weakly stable if there is no pair (θ_i, ϕ_j), each of whom is either unmatched in $\mathcal{M}$ and the other appears in his/her list, or strictly prefers the other to his/her partner in $\mathcal{M}$. A weakly stable matching can be found by arbitrarily ordering all ties, but the ways in which ties are broken affect the solution. It would be desirable finding a maximum cardinality weakly stable marriage, i.e. to maximize the number of couples, but it turns out to be a NP-hard problem, even under strong restrictions.

B. The Revised Stable Marriage Solution

In this section, we will introduce the SMTI Revised Algorithm, which we have developed to solve the task assignment problem in camera networks. It is inspired by SMTI Problem, where agents play the role of men and tasks that of women, but it presents some differences due to the dynamics of the problem and the difficulties of finding a stable and optimum solution.

At every iteration of the SMTI Revised Algorithm, each agent a_i gives every $\tau_j \in \mathcal{T}(a_i)$ a profit score computed as:

$$c_j(t) = \alpha \cdot \mathrm{pr}(j) + \frac{1-\alpha}{T_{drop}} \cdot (t - T_{occ}(j)), \qquad (13)$$

where $c_j(t)$ is the profit vector use in Eqn. (5), $\mathrm{pr}(j)$ is the intrinsic priority of the task τ_j, $(t - T_{occ}(j))$ is the lifetime of the task; $\alpha \in (0, 1]$ allows balancing between the absolute priority term and the obsolescence driven one: $\alpha \to 0$ gives more importance to tasks whose life span is close to drop time; $\alpha \to 1$ weighs more tasks equipped with higher intrinsic priorities. Note that in between a new task arrival or task completion, the different $c_j(t)$'s grow with the same rate, as graphically illustrated in Fig. 3, therefore the optimal assignment does not vary between these events and the $c_j(t)$'s can be evaluated only when a new task appear or an old task is completed.

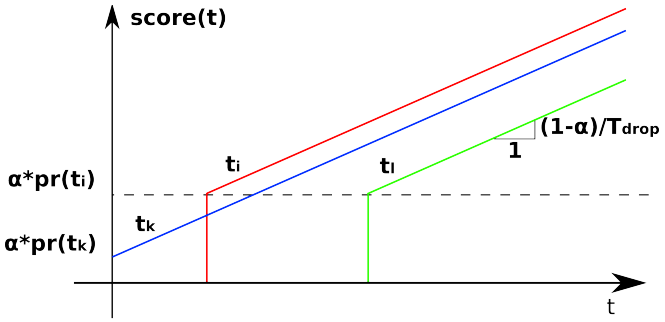


Fig. 3: Profit score as a function of time for a specific target sequence.

Each agent a_i builds its own preference list containing all the elements $\tau_j \in \mathcal{T}(a_i)$, sorted by the scores c_j's defined above. This preference list can be incomplete since it contains only the feasible tasks $\mathcal{T}(a_i)$, and may contain ties since two or more tasks can have received the same score c_j from one agent. So this problem can be thought as a SMTI, but here we will limit ourselves to find a stable (not necessarily optimum) match, in order to avoid the difficulties related to NP-hard complexity. However, differently from the standard SMTI where a fixed arbitrary order is imposed on the agents' preference lists to break the ties, here we propose a different strategy that dynamically favor the assignment of tasks to avoid idle agents. More precisely, each task has its own preference list of agents based on how loaded the agents are and on the position of the task in the agents' preference list. This list is used to determine when a task swap from one agent to another is convenient. This heuristic has no optimality guarantee but maintain stability and termination in finite time, and exhibits good performance in simulations as shown in Sec. VI.

The proposed approach is summarized in Algorithm 1. Lines 1-6 correspond to the algorithm initialization. Line 8 states that the order from which agents are selected is arbitrary since it does not affect stability nor termination of the algorithm (but it does affect final assignment), thus being suitable for decentralized implementation. Line 10 is one of the difference of our algorithm from standard SMTI, since our algorithm each agent starts evaluating its preference list at every iteration, while in latter the task that are evaluated at one iteration are eliminated form the list. This modification allows the algorithm to be adaptive to new task arrival or task completion. Also Line 13 give rise to a substantial difference between our algorithm and the standard SMTI solution. In fact in the latter an arbitrary order is set for the agent preference list and then each task is sequentially evaluated for potential swapping, while here we first check if there is any unmatched task in a tie block, and only if this condition is not satisfied we then evaluate tasks for potential swapping. This apparently innocuous difference allows better

load balancing among the agents, since the algorithm does not steal a task from another agent if there is an unassigned task with the same profit in its list.

Algorithm 1 SMTI-Revised Algorithm

Require: new task occurrence or task completion.
for all agent a_i **do**
  compute ordered task list for agent a_i based on c_j.
  group tasks of a_i with same score (ties) in blocks b_ℓ
end for
for all task τ_j **do**
  compute ordered agent list for task τ_j based on u_{ij}.
end for
repeat
  for all agent a_i *(arbitrary order)* **do**
    for all block b_ℓ *(decreasing score ordered)* **do**
      if agent a_i is running any task in the block b_ℓ **then**
        Skip to next agent.
      else if there exists any unassigned task $\tau_j \in b_\ell$ **then**
        Assign task τ_j with lower ID to agent a_i.
        Skip to next agent.
      else
        for all task $\tau_j \in b$ *(increas. ID ordered)* **do**
          if swap is strictly convenient **then**
            Assign task τ_j to agent a_i.
            Skip to next agent.
          end if
        end for
      end if
    end for
  end for
until no more changes in assignment occur.

C. Task preference list and swap policy

To begin with, it is important to point out that the adopted swap policy is based on the women (the tasks in our context) preference list as in the classical Gale-Shapley Algorithm. Let τ_j be a task and $\mathcal{A}(\tau_j)$ the set of agents a_i compatible with τ_j. The binary vector u_{ij} is defined for each $a_i \in \mathcal{A}(t)$ as $u_{ij} = (r_{ij}, \ell_i)$, where r_{ij} is equal to the number of tasks that follow τ_j in a_i's preference list, and ℓ_i is the length of the a_i's preference list. Then the task τ_j builds its own preference list of the agents based on $u_{ij}, \forall a_i \in \mathcal{A}(\tau_j)$ as follows: a_i has higher preference than a_k for task τ_j if $u_{ij} < u_{kj}$[3]. In the event of a tie, i.e. $u_{ij} = u_{kj}$, then the agent ID is used to order them. Based on this list, a swap of task τ_j from agent a_i to agent a_k is strictly convenient if a_k appear before a_i in the preference list of task τ_j.

D. Comparison with Other Algorithms

SMTI Revised Algorithm will be compared with other approaches which are going to be introduced:

[3]The ordering is based on the rule $u_{ij} < u_{kj}$ if $r_{ij} < r_{kj}$ or if $r_{ij} = r_{kj} \wedge \ell_i < \ell_k$.

- Centralized Assignment or PLI (Linear Integer Programming): this approach provides the best solution with regard to the maximization of the utility function defined in Eqn. (5), where $c_j(t)$ are defined in Eqn. (13). It can be solved by means of the Simplex Algorithm.
- Naive Approach: each agent which is unload or is patroling an area randomly picks an available unmatched task (different from patroling); the other agents keep executing their current tasks.
- Greedy Approach: each agent $a_i \in \mathcal{A}$ scans the task pool for the best (in terms of highest profit $c_j(t)$) unmatched task τ_{best}: if it is better than the task a_i is currently performing, a_i starts executing τ_{best}, otherwise a_i keeps executing the old task.

VI. Simulations

Simulations are run on an $8 agents \times 9 areas$ scenario ($n = 8$, $p = 9$), with a feasible task load $C \simeq 0.48$, where C is the rate of incoming task per single agent multiplied by the average time for an agent to complete the task. Results are grouped in terms of the algorithm type and are compared for different value of parameters T_{drop} and α.

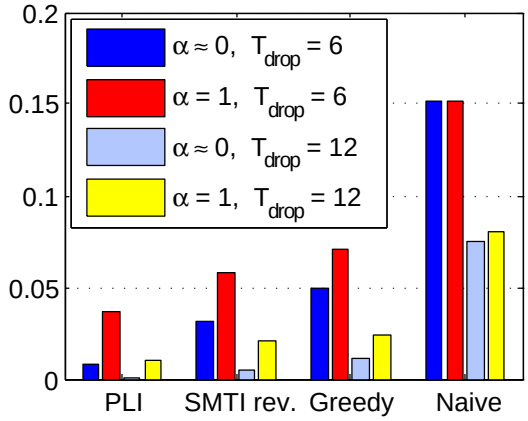
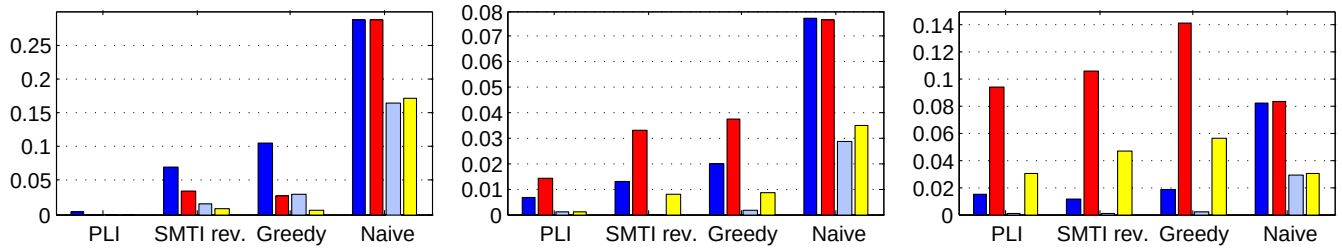


Fig. 4: Drop rate.

To begin with, we take into consideration the performances in terms of dropped tasks according to Eqn.(12). The plot in Fig. 4 shows the importance of the choice of the parameter T_{drop}. As expected, the longer it is, the smaller is the dropping rate. The maximum value it can assume is determined by the practical implementation, as it describes the maximum allowed duration of a task life. After T_{drop}, a task is to be considered obsolete, so it is removed from the pool.

Still considering the dropping rate, it is interesting to compare the behavior of the different types of tasks, depending on the values of parameter α. Fig. 5 shows how the highest priority tasks are preferred when $\alpha = 1$. This occurs because α weighs the intrinsic priority of the tasks. On the one hand, since this is maximum for considered tasks, PLI, SMTI Revised and Greedy (that make use of the utility function described in Eqn. (5) show extremely low dropping rate for this kind of tasks. On the other hand, lower intrinsic priority tasks are penalized. The latter kind of tasks, that have the



(a) *High-priority tasks* (b) *Mid-priority tasks* (c) *Low-priority tasks*

Fig. 5: Drop rate for different task priorities (color legend as in Fig. 4).

smallest priority degree, show the worst dropping rate for high values of α.

As regards to the average waiting time of the executed tasks given by Eqn. (11), the overall effect of choosing high values of α is to reduce it, as shown in Fig. 6. This happens because $1 - \alpha$ weighs the task life span, giving preference to tasks that are close to be dropped. As a consequence, the time spent in queue by the executed tasks results longer, in average. However, observing in detail the task behavior, shown in Fig. 7, it can be found that low-priority tasks do not follow the general trend.

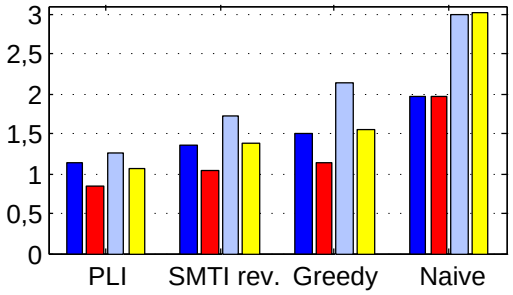


Fig. 6: Average waiting time (color legend as in Fig. 4).

In order to achieve a better comprehension of these results, it is useful to think about the role of α. Considering the array of all tasks (both active and waiting), sorted by occurrence time, we can say that T_{drop} defines the length of this structure. The time interval $\mathcal{I}$ in which the probability of selecting a task is higher, is linked to α. The bigger it is, the wider $\mathcal{I}$ is (over time). Viceversa, for lower values of α, $\mathcal{I}$ shrinks to its fixed extreme $t - T_{\text{drop}}$, t being the current time. Dually, α plays the same role with regard to priority: higher values of α imply bigger probability of selecting only the highest priority tasks. Fig. 8 gives a graphic representation of the role of α.

In spite of the fact that high values of α generally cause the waiting time to decrease, low-priority tasks behave differently. This is because they have the lowest intrinsic priority, so with $\alpha \approx 1$ agents tend to neglect them. Obviously, longer T_{drop} allows the queue to grow, so waiting time are increased.

We have already stated that we are interested in a good trade-off between continuity and optimality. We now consider the first one w.r.t. Eqn. (10), comparing the different candidate algorithms shown in Fig. 9. As expected, the

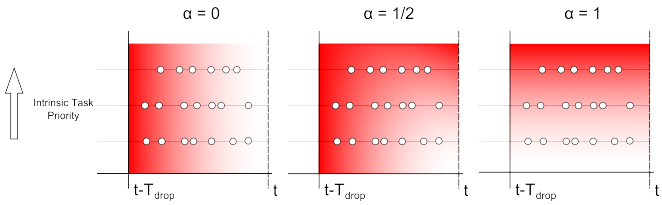


Fig. 8: Sliding pool of tasks. Color intensity is proportional to associated profit score $c_j(t)$.

PLI algorithm does not exhibit the desired continuity. This happens because this approach is memoryless: it does not take into account the previous matching in order to update the assignment when a new task occurs or is completed. Greedy and Naive show the best results, since they are designed to be extremely conservative in matching tasks and agents. The best performances are exhibited by the Greedy algorithm when $\alpha \approx 0$, because in such schenario agents evaluate tasks considering only their occurrence time. As a consequence the oldest tasks are always at the top of the agents' lists. SMTI Revised achieves good performances, thanks to the fact that swaps are limited by restrictive conditions based on u_{ij}.

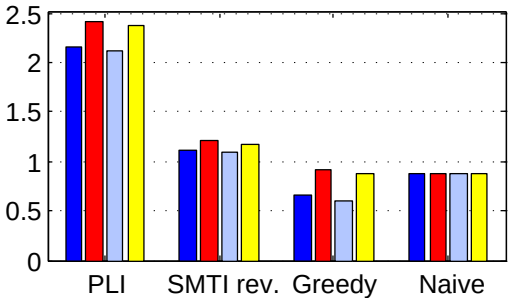
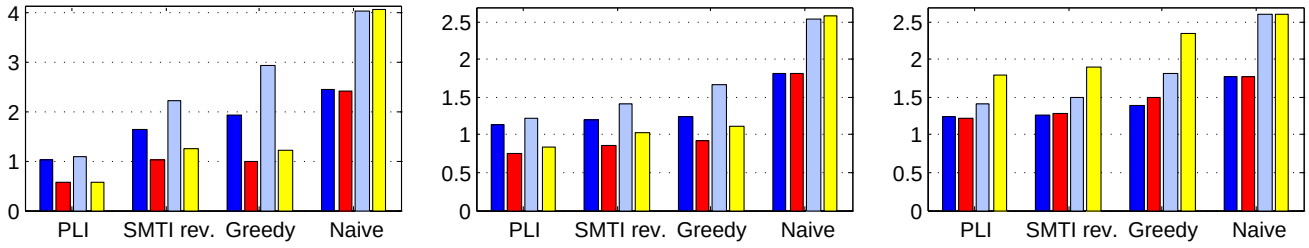


Fig. 9: Discontinuities in task execution (color legend as in Fig. 4).

Another meaningful aspect that has to be inspected is the presence of undesired idle agents in terms of Eqn. (9) shown in Fig. 10. Under our assumptions we can exclude an instantaneous optimal solution in which some agents are unloaded. As matter of fact, PLI never shows idle agents. SMTI Revised achieves good performances since it wilfully tries to match each agent. In order to do it, in comparing agents to be assigned to a task, it weighs the shortness of their list of unexplored feasible tasks, i.e. r_{ij} used for



(a) *High-priority tasks* (b) *Mid-priority tasks* (c) *Low-priority tasks*

Fig. 7: Average waiting time for different task priorities (color legend as in Fig. 4).

the swap. Greedy and Naive do not guarantee the idle state avoidance. As a general trend, when working with a longer T_{drop}, the pool contains more elements and makes unlikely that an agent has no tasks to perform.

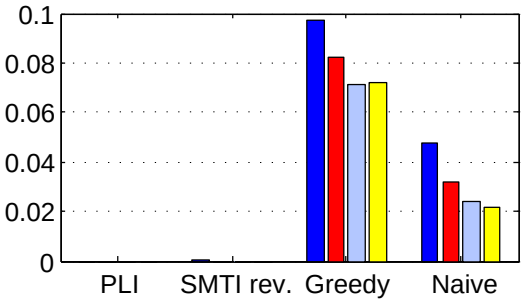


Fig. 10: Idle state occurrence rate (color legend as in Fig. 4).

Referring to optimality, a remarkable aspect is the relative intrinsic priorities of assigned tasks, as defined in Eqn. (8) and shown in Fig. 11. The best results are obtained by PLI and SMTI Revised and there are not significant variations with different values of α.

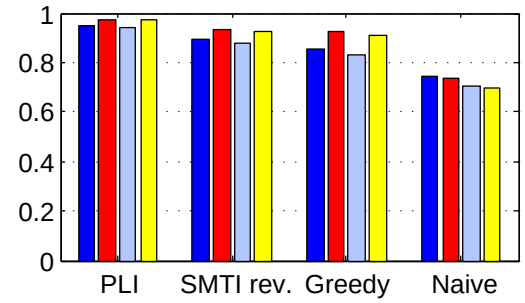


Fig. 11: Relative assigned priorities (color legend as in Fig. 4).

VII. Conclusion

The proposed SMTI Revised algorithm shows good performances. In maximizing assigned priorities, avoiding idle agents and reducing both drop rate and average waiting time, it is similar to the centralized assignment (PLI). Moreover, it is better in terms of continuity. As previously discussed, SMTI Revised can be easily turned into a distributed algorithm (with local interactions only), even though the costs in terms of both communication and code implementation should be further analyzed. As regards the PLI algorithm, despite the better performances, it is hardly useful over a real world scenario, due to the bad scalability and the unacceptable discontinuity in task execution.

A point that will be studied next is the problem of keeping the task pool up to date when algorithms are implemented in distributed form. In this context, issues regarding the asynchronous management of the agent task list and occurrence/dropping of tasks will be of interest. On the other hand, the possibility of scaling up in a fully distributed fashion for wide area networks, where the local feature of both information sharing and agent memory content is enhanced, will be in focus.

References

[1] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE, special issue*, vol. 95, no. 1, pp. 138–162, January 2007.

[2] N.Michael, M. M.Zavlanos, V.Kumar, and G.J.Pappas, "Distributed multi-robot task assignment and formation control," in *Proc. of the 3rd IEEE International Conference on Robotics and Automation, ICRA2008*, 2008.

[3] B.J.Moore and K.M.Passino, "Distributed task assignment for mobile agents," *IEEE Transactions on automatic control*, vol. 52, no. 4, pp. 749–753, 2007.

[4] L. Brunet, "Consensus-based auctions for decentralized task assignment," Master Thesis, Massachusetts Institute of Technology, 2008.

[5] M.J.Feiler, "On distributed search in an uncertain environment," in *Proc. of the 1st IFAC Workshop on Estimation and Control of Networked Systems, Venice, Italy*, 2009.

[6] B.Lagesse, "A game-theoretical model for task assignment in project management," in *In IEEE International Conference on Management of Innovation and Technology*, 2006.

[7] W.Hunt, "The stable marriage problem," [online] http://www.csee.wvu.

[8] D.F.Manlovea, R.W.Irvinga, K.Iwama, S.Miyazaki, and Y.Morita, "Hard variants of stable marriage," *Theoretical Computer Science*, vol. 276, pp. 261–279, 2002.

[9] D.Gale and L. S.Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.