European Journal of Operational Research 260 (2017) 725–738

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

The block-information-sharing strategy for task allocation: A case study for structure assembly with aerial robots☆

L. E. Caraballo[a], J. M. Díaz-Báñez[a,*], I. Maza[b], A. Ollero[b]

[a] Department of Applied Mathematics II, University of Seville, Spain
[b] Robotics, Vision and Control Research Group, University of Seville, Spain

ARTICLE INFO

Article history:
Received 23 December 2015
Accepted 30 December 2016
Available online 3 January 2017

Keywords:
Multi-agent systems
Dynamic task allocation
Distributed algorithm
Assembly line balancing

ABSTRACT

A new paradigm for task allocation in cooperative multi-robot systems is proposed in this paper. The block-information-sharing (BIS) strategy is a fully distributed approach, where robots dynamically allocate their tasks following the principle of *share* & *divide* to maintain an optimal allocation according to their capabilities. Prior studies on multi-robot information sharing strategies do not formally address the proof of convergence to the optimal allocation, nor its robustness to dynamic changes in the execution of the global task. The BIS strategy is introduced in a general framework and the convergence to the optimal allocation is theoretically proved. As an illustration of the approach, the strategy is applied to the automatic construction of truss structures with aerial robots. In order to demonstrate the benefits of the strategy, algorithms and simulations are presented for a team of heterogeneous robots that can dynamically reallocate tasks during the execution of a mission.

1. Introduction

Multi-robot task allocation in dynamic environments is one of the fundamental problems in cooperative robotics and a challenging area in operational research (Dias, Zlot, Kalra, & Stentz, 2006; Gerkey & Matarić, 2004; Korsah, Stentz, & Dias, 2013; Shima, Rasmussen, Sparks, & Passino, 2006; Tsalatsanis, Yalcin, & Valavanis, 2012): given a group of cooperative robots, a global task to be performed in a dynamic environment and a cost function, how should subtasks be allocated to the robots in order to complete the task while minimizing costs?

Let us consider a task such as the cooperative manipulation of structures addressed in the ARCAS European Project (http://www.arcas-project.eu/) funded by the European Commission. One of the goals of this project is to assemble a structure using a team of aerial robots equipped with on-board manipulators. The practical interest of this system can be found in situations where it is required to build a structure in places with difficult access through conventional means (see Fig. 1). The use of aerial robots allows to perform assembly operations in any point in space, which in areas of difficult access represents a relevant advantage over ground robots.

Assembly planning (Ghandi & Masehian, 2015) is the process of creating a detailed plan to craft a whole product from separate parts by taking into account the geometry of the final structure, available resources to manufacture the product, fixture design, feeder and tool descriptions, etc. Efficient assembly plans can significantly reduce time and costs. The assembly planning problem has been shown to be NP-complete (Kavraki, Latombe, & Wilson, 1993) and covers three main assembly subproblems: sequence planning, line balancing, and path planning.

Jimenez (2011) presents a classification of structures according to different features: number of hands, monotonicity (whether operations of intermediate placement of subassemblies are required), linearity (whether all assembly operations involve the insertion of a single part or multiple parts which have to be inserted simultaneously), and coherence (whether each part that is inserted will touch some other previously placed part). The structures considered in this paper are sequential (for two robots), monotone, linear and contact-coherent.

This paper focuses on the line balancing stage and introduces a novel paradigm for coordinating multiple robots in the execution of cooperative tasks in dynamic scenarios. The basic idea is to share information within a group or *block* of robots before assigning subtasks to all the members of the block. In this way, the method simulates a centralized system using a decentralized approach in which the robots share information in order to guarantee

☆ This work has been partially supported by the projects MTM2016-76272-R funded by the Spanish Government and ARCAS(FP7-ICT-287617) funded by the EU FP7.

* Corresponding author.
E-mail address: dbanez@us.es (J.M. Díaz-Báñez).

http://dx.doi.org/10.1016/j.ejor.2016.12.049
0377-2217/

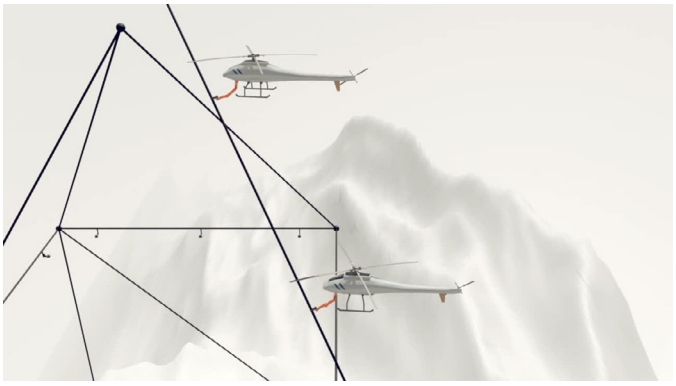

Fig. 1. Two aerial robots equipped with LWR KUKA robotic arms manipulating a bar to build a structure in places with difficult access by conventional means.

a local optimal solution in each stage of the algorithm. Our strategy fits the *locally centralized paradigm* mentioned in Cao, Fukunaga, and Kahng (1997) and allows the team to perform task allocation periodically in an entirely decentralized manner. Moreover, by allocating tasks to robots repeatedly, a team can adapt as circumstances change achieving fluid coordination.

The block-information-sharing (BIS) strategy was introduced in Caraballo et al. (2014) by the authors for area monitoring missions as a generalization of the one-to-one paradigm presented in Acevedo et al. (2013a, 2014). In these papers, the authors have experimentally demonstrated that this strategy converges to an optimal situation in which a particular objective function, the idle time function, is minimized. However, neither theoretical proofs on the convergence nor approach formalization have been published so far.

In this paper, the convergence of the BIS strategy is formally proved in a general task allocation scenario. Moreover, it is shown how to use this strategy to design a fault-tolerant approach for structure construction using a cooperative team of aerial robots. The robots work in parallel and the dynamic assignments of the tasks are performed using blocks in order to maintain a balanced allocation where each aerial robot approximately spends the same time to construct the assigned section. Thus, the maximum time a robot spends to complete the assigned section of the construction is minimized. Consequently, the benefit obtained is twofold: the total time for the construction is optimized and the robustness of the approach is guaranteed since the system is periodically rebalanced to cope with certain events, such as battery failure or the loss of some aerial robots, during the construction process.

The paper is organized as follows. The optimization problem is formulated in Section 3. In Section 4 the block-information-sharing paradigm for a general scenario is formally introduced, proving the convergence to the optimal allocation. In a case study presented in Section 5, the paradigm is employed in a structure assembly task, where the proposed approach is implemented and validated in a simulation of the construction of a structure using a team of aerial robots equipped with on-board manipulators. The obtained results of these simulations are shown at the end of Section 5. Finally, Section 6 closes the paper with some conclusions.

2. Related work

The task allocation problem has been widely studied in different areas. Centralized mechanisms have advantages including a guaranteed optimal solution. However, their limitations are well known, i.e. their slow response to dynamic events and vulnerability to failures. In dynamic and uncertain environments, decentralized mechanisms are preferred. Decentralized task allocation approaches for autonomous agents have been well studied in robotics (Dias et al., 2006; Gerkey & Matarić, 2004; Khamis, Hussein, & Elmogy, 2015; Korsah et al., 2013; Lemaire, Alami, & Lacroix, 2004). Gerkey and Matarić (2002) presents a dynamic task allocation scheme using an auction process for a heterogeneous robot team. Dias (2004) introduces a market-based multi-robot task allocation architecture in which robots are modeled as self-interested agents with the goal of maximizing individual profits. In practice, robots have a limited communication capability which may influence development and performance of the task allocation algorithm significantly.

In addition, our problem is also related to the well-known assembly line balancing (ALB) optimization problem in operational research (Boysen, Fliedner, & Scholl, 2007; Scholl & Becker, 2006). The ALB problem deals with partitioning the total assembly operations into a set of m elementary tasks $o_i (i = 1, \ldots, m)$ with times t_i, and assigning them to a team of n assembly robots $r_k (k = 1, \ldots, n)$ such that all robots approximately spend equal assembly times and the so-called "precedence constraints" between operations are satisfied. Assuming that the set S_k of tasks is assigned to the kth robot, its assembly time is $t(S_k) = \sum_{j \in S_k} t_j$. The ALB problem is NP-hard (Scholl & Becker, 2006) and soft computing approaches have been proposed in Rashid, Hutabarat, and Tiwari (2012). The main goal in an ALB model is the allocation of the tasks among stations so that the precedence relations are not violated and a given objective function is optimized. Besides balancing a newly designed assembly line, an existing assembly line has to be re-balanced periodically or after certain changes in the production process or the production plan. In our case, the team of cooperative robots must satisfy the precedence constrains between the construction operations and therefore the performance of the system is related to the time a robot is waiting for other neighbors to continue the construction. Because of the long-term effect of balancing decisions, the objective functions have to be carefully chosen while considering strategic goals. In structure construction using aerial robots, the total time for the construction is improved when the timeouts are minimized.

3. Problem statement

Consider a cooperative team of autonomous and heterogeneous robotic agents and a task which has to be accomplished by the team. In a cooperative framework, it is assumed that the task is *divisible* and *parallelizable* in such a way that it can be partitioned so that each subtask is assigned to a robot in the team. Individual agents execute their subtasks independently except in the common parts where neighbors need to coordinate their cooperation.

The time to perform the overall task is determined by the maximum time spent by the individual robots on their subtasks while a balanced allocation needs to be maintained in order to prevent overload and ensure efficiency. Considering the parallel nature of this scenario, performing the task in the shortest possible time requires minimizing the maximum time spent by the individual robots.

As in the ALB problems, the task to perform can be modeled as a set of operations where subtasks correspond to a disjoint partition of the task. The formalization of the problem is illustrated with two examples:

(i) building a structure using assembly parts (we assume equal complexity of placing the parts) in which the task is the set of assembly operations;
(ii) monitoring a region, where the task is the set of all points in the region.

Let T be the set representing the general task to be performed, where each subtask is a subset of T. Let $U_1, U_2, \ldots, U_n$ be a team

of n robots to perform the task. The *capability coefficient* indicates the amount of operations which can be performed by a robot in a given time frame. Let $c_i \in \mathbb{R}$ denote the capability coefficient of U_i to perform a task. For scenario (i), the capability coefficient of a robot is the number of assembly operations that can be performed per unit of time; for scenario (ii) it indicates the area which can be monitored in one unit of time. Let $\mu : \Sigma \to \mathbb{R}$ be a function to measure the task T, where Σ is a σ-algebra on T. For example, for a subtask $T' \subseteq T$, in (i) $\mu(T')$ is the number of assembly operations in T' and in (ii) $\mu(T')$ is the area of the region T'. The time required by the robot U_i to perform a subtask T' is given by

$$\frac{\mu(T')}{c_i}.$$

Let T_i be the subtask assigned to the robot U_i. Our goal is to obtain a partition $\{T_1, \ldots, T_n\}$ of T such that

$$\max_i \left\{ \frac{\mu(T_i)}{c_i} \right\}$$

is minimized. Using the definitions above, the key issue addressed in this paper is the following:

Problem 3.1 (Efficient task allocation)**.** Given a heterogeneous team of robots $U_1, U_2, \ldots, U_n$ and a task T,

$$\begin{aligned} \textit{Minimize} \quad & f(T_1, T_2, \ldots, T_n) = \max_i \left\{ \frac{\mu(T_i)}{c_i} \right\} \\ s.t. \quad & \bigcup_{i=1}^{n} T_i = T, \\ & T_i \cap T_j = \emptyset, \quad \forall i \neq j. \end{aligned}$$

If the general task T is a discrete set of atomic (indivisible, basic) operations, that is, $T = \{a_1, a_2, \ldots, a_m\}$, then for a subtask $T_i \subseteq T$, it follows that

$$\mu(T_i) = \sum_{a_j \in T_i} \mu(\{a_j\}) = \sum_{j=1}^{m} \mu(\{a_j\}) x_{ij}$$

where $x_{ij} = 1$ if $a_j \in T_i$, and $x_{ij} = 0$ otherwise. Our problem can be rewritten as follows:

$$\textit{Minimize} \quad f(x_{11}, \ldots, x_{1m}, \ldots, x_{n1}, \ldots, x_{nm}) = \max_i \left\{ \frac{\sum_{j=1}^{m} \mu(\{a_j\}) x_{ij}}{c_i} \right\} \tag{1}$$

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

Note that a generalization of example (i), where the complexity of placing pieces varies and c_i is the capability of the i-th robot to manipulate pieces, can be also modeled using statement (1).

It is easy to see that in an optimal allocation all robots spend the same amount of time to perform their respective subtasks according to their capabilities. Consequently, to ensure an optimal task allocation, it follows that:

$$\mu(T_i) = c_i \frac{\mu(T)}{\sum_{j=1}^{n} c_j}. \tag{2}$$

In this paper we are interested in a fault-tolerant and decentralized strategy to solve Problem 3.1 for multiple tasks. On the one hand, the conditions of the environment as well as the capabilities of the robots can change dynamically, requiring an update of the task allocation. On the other hand, in many real situations, robots work in an inaccessible areas and it is thus desirable to perform the tasks in a decentralized manner due to communication range limitations. Therefore, a decentralized strategy is required to recover and keep an optimal task allocation during the execution of a mission despite dynamical changes in the team or the environment. In the following section, the BIS strategy is presented as a solution to this problem.

In the remainder of this paper, it is assumed that the global task to be performed can be represented as a measurable set T, and every subtask is a measurable subset T_i of T. Also, in order to simplify the notation, the values of $\mu(T)$ and $\mu(T_i)$ are referred to as T and T_i, respectively.

4. The block-information-sharing paradigm in dynamic task allocation problems

The so-called *one-to-one strategy* (Acevedo et al., 2013a; 2014), which was initially applied to area monitoring missions, can be easily adapted to general tasks. Suppose that our team has a non-optimal task allocation which we wish to turn into an optimal one in a decentralized manner without any intervention of external information sources. The key idea behind the one-to-one strategy is to share the information on capabilities and subtasks among neighboring robots so they can re-allocate subtasks between them. It has been experimentally demonstrated that the task allocation in the system converges to an optimal partition of the general task when this process is repeated iteratively. Unfortunately, the convergence may be too slow in some cases. In Caraballo et al. (2014), examples of configurations where the one-to-one strategy converges only slowly to an optimal partition are shown and the BIS strategy is introduced as a generalization of the one-to-one approach that allows to accelerate the convergence to an optimal solution. In this paper, we lift the BIS strategy on a higher level of abstraction and present the approach as a general paradigm with multiple applications in cooperative robot systems.

4.1. General strategy

Consider a general task to be executed by a cooperative team of heterogeneous robots equipped with wireless communication for information sharing. Furthermore, assume that the environment conditions as well as the individual performance of the robots change dynamically affecting the global performance of the team. We are interested in a strategy to obtain and maintain a balanced subtask allocation which allows to complete the global task in minimum time. As the communication range is limited, it is assumed that each robot can only share information with its neighbors. This assumption implies a *communication graph*: Nodes correspond to robots and an edge between to nodes exists only if the respective robots are able to share information. To ensure that all members of the team are involved in the strategy, the communication graph needs to be connected. In Fig. 2a two-dimensional communication graph is illustrated (in real-world applications with aerial robots, a three-dimensional graph can be considered).

We define a *block* as a connected subgraph of the communication graph which we denote with the indices of the robots within the block. For example, $B = \{1, 3, 4\}$ denotes the block composed by the robots U_1, U_3 and U_4. A partition of the communication graph into blocks is a *block-configuration*. The BIS strategy supports two or more block-configurations such that their union contains all edges of the communication graph. Fig. 3 shows two such block-configurations which, since their union contains all edges, can be used in the strategy.

The BIS strategy is based on the organization of a team of robots according to a block-configuration, where, within a block,

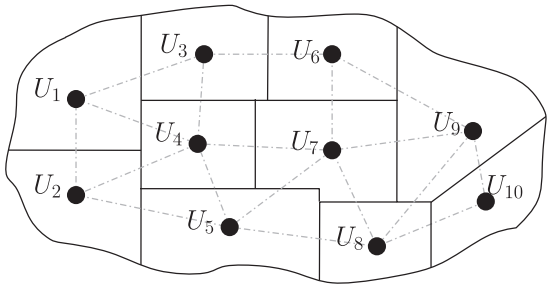

Fig. 2. Representation of an area allocation among robots and the associated communication graph.

robots share information among each other. Before creating an allocation, the robots remain in their position until all information about capabilities and tasks has reached all of the robots in the block. The iterative approach is as follows: When the team completes a task allocation using a block-configuration, another block-configuration is used in the next stage. The reallocation process in a block uses Eq. (2) locally. Formally, if B is a block and U_i is a robot in B, the new subtask T_i' for U_i is given by

$$T_i' = c_i \frac{\sum_{j\in B} T_j}{\sum_{j\in B} c_j}. \tag{3}$$

An example is given in Fig. 3: Suppose that the robots start in the configuration shown in Fig. 3a with partition into blocks {1, 2}, {3, 4, 5, 6, 7, 8} and {9, 10}. Within each block, robots share information and reallocate their subtasks. After this first step, each block has a task allocation that is locally optimal. In the next stage, the team considers the configuration shown in Fig. 3b composed of the blocks {1, 2, 3, 4, 5} and {6, 7, 8, 9, 10} and reallocates the tasks in these blocks correctly. After that, the team returns to the previous configuration in Fig. 3a and so on.

In the next subsection, we formally prove that the BIS strategy, which takes advantage of local information to correctly distribute tasks, converges to an optimal task allocation for Problem 3.1. We aim to minimize the maximum time required by a robot to perform its subtask. In this scenario, which corresponds to an optimal solution, all heterogeneous robots spent an equal amount of time. The BIS strategy tends to produce such an allocation and ensures good performance for setups for which a balanced allocation is optimal. Consequently, this paradigm can be applied to multiple scenarios where a task is executed by cooperative robots. It allows to design efficient decentralized algorithms to optimize different objective functions, for instance, the maximum idle time in monitoring or surveillance missions (Acevedo et al., 2013a, 2014).

We furthermore identify the possibility of applying this scheme (sharing information and taking decisions in blocks while alternating block-configurations) in alternative scenarios where any type of information (which is not necessarily related to task allocation) is propagated in order to optimize the global system performance or a specific cost function (other than the robots spending an equal amount of time to perform their subtasks).

4.2. Convergence proof

The following conditions are assumed: the general task is divisible and parallelizable, the communication graph is connected and the union of the selected block-configurations is a covering of the set of edges of the communication graph.

The BIS strategy is an iterative approach where each iteration corresponds to a different block-configuration of the team. Let us represent by $T_r^{(j)}$ the subtask assigned to the robot U_r in the jth iteration. If U_r belongs to block B in the jth iteration then Eq. (3) is written as

$$T_r^{(j)} = c_r \frac{\sum_{i\in B} T_i^{(j-1)}}{\sum_{i\in B} c_i}.$$

In the following results we use the values:

$$M^{(j)} = \max_i \left\{ \frac{T_i^{(j)}}{c_i} \right\} \quad \text{and} \quad m^{(j)} = \min_i \left\{ \frac{T_i^{(j)}}{c_i} \right\},$$

with $i \in \{1, \ldots, n\}$.

Lemma 4.1. *For all $k \geq 0$ it follows that*

$$m^{(j)} \leq \frac{T_i^{(j+k)}}{c_i} \leq M^{(j)} \quad \forall i \in \{1, \ldots, n\}.$$

Proof. We proceed to use mathematical induction in k. If $k = 0$, then $m^{(j)} \leq \frac{T_i^{(j+k)}}{c_i} = \frac{T_i^{(j)}}{c_i} \leq M^{(j)}$ by definition of $M^{(j)}$ and $m^{(j)}$. Assume as an induction hypothesis that for a fixed value k the claim is fulfilled, $m^{(j)} \leq \frac{T_i^{(j+k)}}{c_i} \leq M^{(j)}$. Let U_r be an arbitrary robot and let B be the block where U_r lies in the $(j+k+1)$th iteration, then:

$$T_r^{(j+k+1)} = c_r \frac{\sum_{i\in B} T_i^{(j+k)}}{\sum_{i\in B} c_i}.$$

From the hypothesis we deduce:

$$\begin{aligned} c_i m^{(j)} &\leq T_i^{(j+k)} \leq c_i M^{(j)} \\ \sum_{i\in B} c_i m^{(j)} &\leq \sum_{i\in B} T_i^{(j+k)} \leq \sum_{i\in B} c_i M^{(j)} \\ m^{(j)} \cdot \sum_{i\in B} c_i &\leq \sum_{i\in B} T_i^{(j+k)} \leq M^{(j)} \cdot \sum_{i\in B} c_i \\ m^{(j)} &\leq \frac{\sum_{i\in B} T_i^{(j+k)}}{\sum_{i\in B} c_i} \leq M^{(j)} \end{aligned}$$

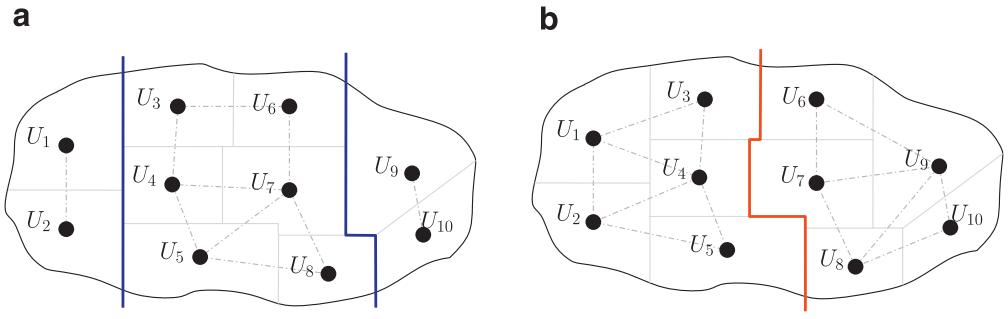

Fig. 3. Block-information-sharing partition samples.

$$c_r m^{(j)} \le c_r \frac{\sum_{i\in B} T_i^{(j+k)}}{\sum_{i\in B} c_i} \le c_r M^{(j)}$$
$$c_r m^{(j)} \le T_r^{(j+k+1)} \le c_r M^{(j)}$$

and the result follows. □

Lemma 4.2. *For all $k, l \ge 0$ it follows that:*

(a) if $m^{(j)} < \frac{T_i^{(j+k)}}{c_i}$ then $m^{(j)} < \frac{T_i^{(j+k+l)}}{c_i}$

(b) if $\frac{T_i^{(j+k)}}{c_i} < M^{(j)}$ then $\frac{T_i^{(j+k+l)}}{c_i} < M^{(j)}$

Proof. Only claim (a) is proven since the proof for claim (b) is analogous. Mathematical induction in l will be used. If $l = 0$, it follows that

$$\frac{T_i^{(j+k+l)}}{c_i} = \frac{T_i^{(j+k)}}{c_i} > m^{(j)}.$$

Suppose as induction hypothesis that $m^{(j)} < \frac{T_i^{(j+k+l)}}{c_i}$ for a fixed value l. Let U_r be an arbitrary robot and let B be the block where U_r lies in the $(j+k+l+1)$th iteration, then:

$$T_r^{(j+k+l+1)} = c_r \frac{\sum_{i\in B} T_i^{(j+k+l)}}{\sum_{i\in B} c_i} > c_r \frac{\sum_{i\in B} c_i m^{(j)}}{\sum_{i\in B} c_i} = c_r \frac{m^{(j)} \sum_{i\in B} c_i}{\sum_{i\in B} c_i} = c_r m^{(j)}$$

and the result follows. □

We are ready to prove the main result of this section.

Theorem 4.3. *The block-information-sharing strategy always converges to an optimal task allocation for Problem 3.1.*

Proof. Let $T_i^{(*)}$ be the task assigned to U_i in an optimal task allocation. Since the time required for every subtask in an optimal task allocation is the same, it is enough to prove that the differences between the times converge to 0. Let $\rho^{(j)}$ be the maximum time difference between the performances of every pair of robots in the jth iteration,

$$\rho^{(j)} = \max_{i,l} \left\{ \left| \frac{T_i^{(j)}}{c_i} - \frac{T_l^{(j)}}{c_l} \right| \right\} = M^{(j)} - m^{(j)}. \tag{4}$$

Note that $\rho^{(j)} \ge 0$ for all j and that the maximum time difference between the performance of every pair of robots in the optimal partition is 0 because all the robots spend the same amount of time to perform their subtasks. We prove that $\rho^{(j)}$ is a decreasing function and then

$$\lim_{j\to\infty} \rho^{(j)} = 0.$$

Suppose that $\rho^{(j)} = c > 0$, we prove that there exists a $q > 0$ such that $\rho^{(j+q)} < \rho^{(j)}$. Let G be the communication graph of the team. Let U_r and U_p be two adjacent nodes in G such that $\frac{T_r^{(j)}}{c_r} = M^{(j)}$ and $\frac{T_p^{(j)}}{c_p} < M^{(j)}$. These two nodes exist since otherwise the graph G would not be connected or $\frac{T_i^{(j)}}{c_i} = M^{(j)}$ for all the nodes and then $\rho^{(j)} = 0$. The union of all the block-configurations contains all the edges of G, so the edge (U_r, U_p) is in some block of a block-configuration. Let $(j+k)$ with $k > 0$ be the index of the first iteration in which the team takes this configuration and U_r and U_p lie in the same block B. Since $\frac{T_p^{(j)}}{c_p} < M^{(j)}$, by Lemma 4.2 we have

$$\frac{T_p^{(j+k-1)}}{c_p} < M^{(j)}. \tag{5}$$

Also, from Lemma 4.1 follows $\frac{T_i^{(j+k-1)}}{c_i} \le M^{(j)}$ for all $i \in \{1, \dots, n\}$, especially for all $i \in B$, then

$$\sum_{i\in B} T_i^{(j+k-1)} \le \sum_{i\in B} c_i M^{(j)}. \tag{6}$$

From $p \in B$, using (5) and (6), follows that:

$$\sum_{i\in B} T_i^{(j+k-1)} < \sum_{i\in B} c_i M^{(j)}, \text{ then dividing by } \sum_{i\in B} c_i$$
$$\frac{\sum_{i\in B} T_i^{(j+k-1)}}{\sum_{i\in B} c_i} < \frac{\sum_{i\in B} c_i M^{(j)}}{\sum_{i\in B} c_i} = M^{(j)}. \tag{7}$$

Multiplying (7) by c_b with $b \in B$ and using Eq. (3) we conclude that

$$T_b^{(j+k)} = c_b \frac{\sum_{i\in B} T_i^{(j+k-1)}}{\sum_{i\in B} c_i} < c_b M^{(j)}.$$

Since $r \in B$ then $T_r^{(j+k)} = c_r \frac{\sum_{i\in B} T_i^{(j+k-1)}}{\sum_{i\in B} c_i} < c_r M^{(j)}$, that is, $\frac{T_r^{(j+k)}}{c_r} < M^{(j)}$. Using the same argument for every robot U_i having $\frac{T_i^{(j)}}{c_i} = M^{(j)}$ we induce that there exists a value $k_M > 0$ such that $\frac{T_i^{(j+k_M)}}{c_i} < M^{(j)}$ for all i.

Analogously we can prove that there exists a value $k_m > 0$ such that $m^{(j)} < \frac{T_i^{(j+k_m)}}{c_i}$ for all i.

Therefore, there exists $q > 0$ ($q = \max\{k_m, k_M\}$) such that for every robot U_i in the team we have $m^{(j)} < \frac{T_i^{(j+q)}}{c_i} < M^{(j)}$, and $\rho^{(j+q)} = M^{(j+q)} - m^{(j+q)} < M^{(j)} - m^{(j)} = \rho^{(j)}$, proving the theorem. □

5. A case study: structure construction

In this section the block-information-sharing paradigm is applied to the construction of a 3D structure using a team of cooperative robots. In line with the objectives of the ARCAS project[1], which considers scenarios involving damaged infrastructure, we have chosen the example of a bridge: The structure is composed of different assembling parts (bars) which must be assembled in a specific order attending to restrictions of union, gravity, etc. (precedence constraints), see Fig. 4. The basic component of the bridge is a cube and the length of a bridge is given by the number of cubes composing it, see Fig. 4.

The parts to be assembled have been deployed (i.e. air-dropped) in specific locations which are referred to as *stores*. Robots should transport parts taken from the stores to the structure site and assemble them correctly. In this scenario, it is more convenient to use aerial robots to avoid obstacles on the ground (also while the structure grows it is becoming an obstacle to the movement of the ground robots). Furthermore, if the structure is multi-level, one bearing another, it is easier for an aerial robot to place part on part in a stack. The aerial robots are equipped with on-board manipulators for the assembly operations.

With the start of the mission, a subtask is assigned to each aerial robot (AR) according to its capabilities. During the mission, a variety of incidents including difficulties while assembling a part, battery degradation, and motor or dexterous manipulator failure, may delay an AR's schedule. In this case, the AR may have to abandon the mission and its subtasks must be reallocated to ensure the completion of the task.

5.1. The problem

Consider a team of ARs equipped with dexterous manipulators to assemble a bridge (one level) as shown in Fig. 4. Each AR has a

[1] http://www.arcas-project.eu

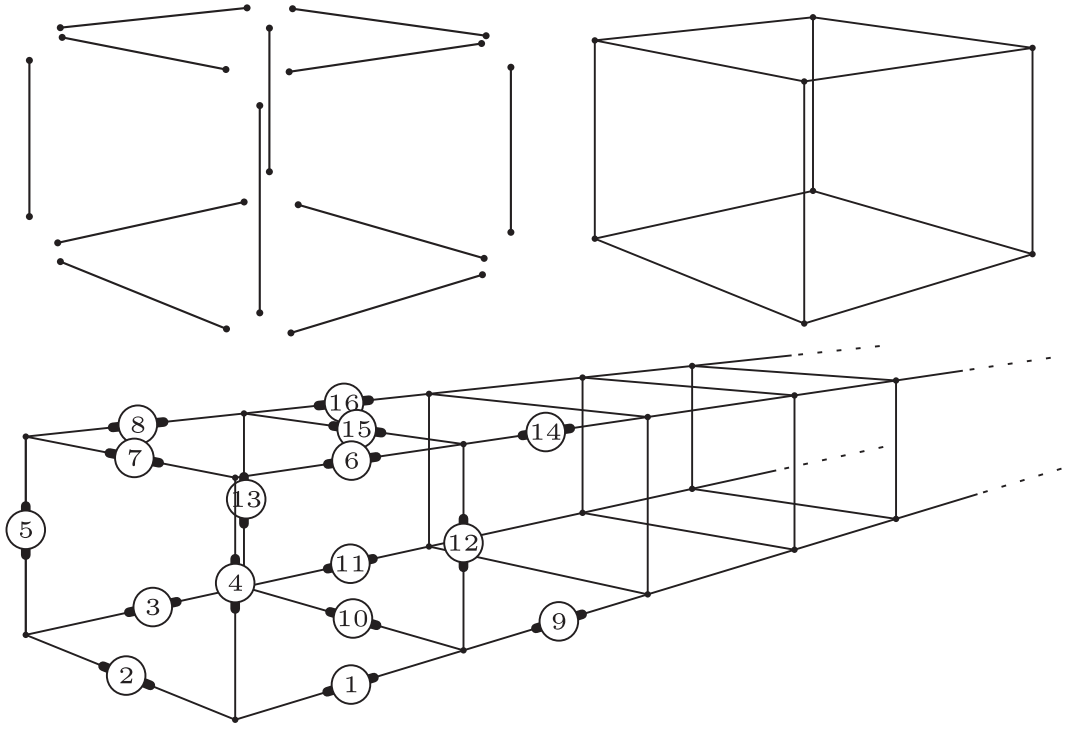

Fig. 4. Bridge structure. The numbers indicate a possible enumeration of the parts.

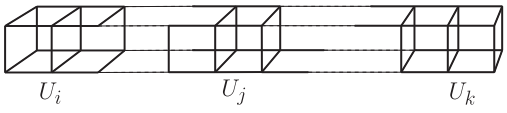

Fig. 5. The bridge is divided into sections assigned to each AR of the team.

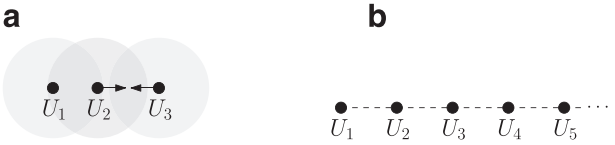

Fig. 6. The communication graph in a line.

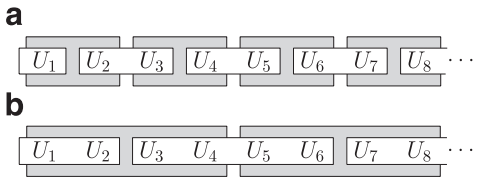

Fig. 7. Block-configurations with blocks of length two (a) and four (b), respectively.

wireless communication device that allows the information interchange with its neighbors and each assembly part has an embedded radio transmitter for its localization and identification. The required sequence of assembly operations has been computed based on assembly-by-disassembly techniques (Jimenez, 2011; Sempere, Llorente, Maza, & Ollero, 2014) in advance. The assembly tasks should be divided into subtasks which can be executed in a parallel manner, to avoid waiting for the placement of all the supporting parts. The robots should also give higher priority to the common parts in order to minimize the waiting time for their neighbors. According to Problem 3.1, the goal is to keep an assembly operations allocation among all available ARs in such a way that the maximum time an AR spends to execute its subtasks is minimized in a fault-tolerant manner, that is, considering that the number of robots and their capabilities can dynamically change.

5.2. The strategy

A bridge is an elongated structure which can be conveniently divided into longitudinal sections, see Fig. 5. This partition generates separated workspaces in which ARs can operate in parallel with low collision risk. When the partition is given, a number of assembly parts is assigned to each AR according to its capabilities. From this partition, the team can be represented by the sequence $U_1, U_2, \ldots, U_n$ where U_i is the AR corresponding to the ith section of the bridge.

We proceed to the design of a decentralized algorithm based on the BIS strategy. First, a sorted list of bridge parts $P = \{p_1, p_2, \ldots, p_m\}$ is computed considering dependencies between them and their positions in the structure from left to right (details of a possible ordering are given in Section 5.4). Every continuous subsequence of P determines a longitudinal section of the bridge (see Fig. 5). Note that if the subsequence $p_j, p_{j+1}, \ldots, p_{j+r}$ corresponds to U_i, then the subsequence corresponds to U_{i+1} is $p_{j+r+1}, \ldots, p_{j+r+s}$.

Since the workspaces are adjacent sections of the bridge, every robot, except the robots working at the end points, has two neighbors. In Fig. 6a, three robots are shown: U_1 and U_2 are sharing information, and U_2 is able to communicate with U_3 if they move closer to each other. In this scenario, the communication graph in the team of ARs is a simple line as shown in Fig. 6b.

A block in this graph is a continuous subsequence of nodes (ARs) composing a line segment. Therefore, a block-configuration corresponds to a list of consecutive segments. We will work with two block-configurations where all the blocks have the same size except the first or last blocks. Fig. 7a shows two block-configurations with block size two, one in gray and the other in white. Note that these block-configurations cover the set of edges entirely and can be used in a BIS strategy to redistribute the

time	U_1	U_2	U_3	U_4	U_5	U_6	U_7
0u	20/1	20/1	20/1	20/1	20/1	20/1	20/1
20u	16/1	16/1	16/1	17/0.5	16/1	16/1	16/1
22u	19/1	19/1	19/1	8/0.5	16/1	16/1	16/1
42u	15/1	15/1	15/1	6/0.5	12/1	12/1	12/1
44u	15/1	15/1	13/1	6/0.5	13/1	13/1	12/1
64u	11/1	11/1	9/1	4/0.5	9/1	9/1	8/1
66u	10/1	10/1	10/1	5/0.5	9/1	9/1	8/1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 8. Example of the evolution of allocated tasks of the team $U_1, U_2, \ldots, U_7$ using the BIS strategy with blocks of size four. Each cell contains information on the robot in the column's header at the time indicated by the row's header. The format of the cells is p/c, where p is the number of pieces assigned to the robot and c is the capability coefficient.

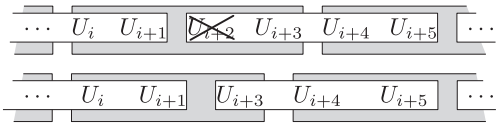

Fig. 9. The block-information-sharing strategy with a failure in U_{i+2}.

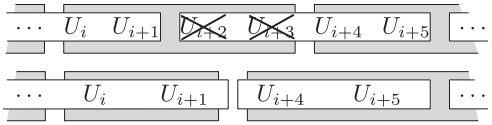

Fig. 10. Block-sharing method with failures in U_{i+2} and U_{i+3}.

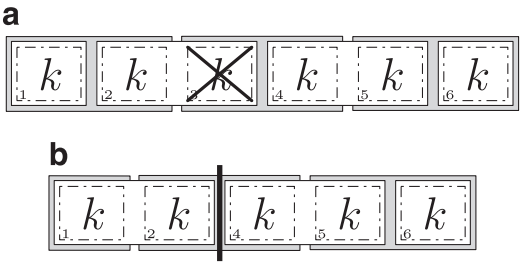

Fig. 11. Recovering of the BIS strategy using $2k$ block size when k consecutive ARs are missing.

workload among the team. Fig. 7b shows a similar case with blocks of size four.

The BIS strategy in this setting works as follows: In certain time intervals, the members of a block share information on placed and unplaced pieces and on their capabilities among them. Once all robots in a block receive this information, a sorted list of unplaced pieces is computed and then a new allocation according to the current capabilities of the available ARs (within the block) is considered. Subsequently, the members of the team change according to the new block-configuration.

To illustrate the strategy, consider the following example: Let $U_1, U_2, \ldots, U_7$ be a team of seven ARs having the same capability of normalized value one. Each team member starts with 20 assembly parts and the time spent by an AR (with capability one) to find and place an assembly part is 5u (five units of time), see Fig. 8. Suppose that, after 20u, the capability of U_4 has decreased to 0.5 and it has assembled 3 pieces while the others have assembled 4 pieces, see Fig. 8. With the new capability, U_4 spends 10u to find and place an assembly part (twice the time spent by others). If the team works without sharing information, the time to complete the structure is the time spent by U_4 to perform its subtask, $20\text{u} + 17 * 10\text{u} = 190\text{u}$. We now demonstrate that this can be improved when the BIS strategy is applied: We assume that the team is using two block-configurations with blocks of size four (see Fig. 7b). The team alternates the block-configurations starting with the gray one. Assume the ARs use an additional 2u to share information and allocate the assembly operations into the block properly. We furthermore assume that team members share information in periodic intervals of 20u. In the interval 20u–22u the team makes a division using the gray block-configuration and in the interval 22u–42u each AR places four parts except U_4 which assembles two parts. After that, in the interval 42u–44u, the ARs update the balanced allocation of the assembly operations applying the white block-configuration and so on. Following this strategy (with no further changes in the team), it becomes obvious that the team completes the structure in 120u. Note that the less time spent on information exchange, the better the overall performance.

Let us now analyze a case where individual ARs fail. Obviously, if the team does not share information, the structure will not be completed if one or more ARs fail. To address this problem in the BIS strategy, the team proceeds as follows: When an AR does not meet a neighbor then it moves in the same direction to meet a new neighbor. When such a meeting occurs (or the AR reaches the end of the bridge), the ARs which should be between the meeting ARs are considered missing. In this way, the remaining robots obtain the new graph representation and assume the tasks of the failed robots. For instance, in Fig. 9 the robot U_{i+2} has failed. When U_{i+1} and U_{i+3} try to meet the right and left neighbors, respectively, they meet each other and consider U_{i+2} as missing. In the example of Fig. 9, if the failure of U_{i+2} is detected using the gray block-configuration, then the other ARs in the block given by U_i, U_{i+1} and U_{i+3} resume the unfinished assembly operations of U_{i+2} in the reallocating process of the task and continue the alternating process between the gray and white configurations.

Unfortunately, the system cannot always recover so easily from failures. Let us consider the same situation mentioned above, but now the AR U_{i+3} also fails. In this new situation, if the ARs try to keep the same block-configuration, the edge (U_{i+1}, U_{i+4}) will be not present in any block (see Fig. 10) and the convergence to an optimal task allocation is not guaranteed.

A generalization of this situation is used to explain how to solve this problem: Assume a team of ARs working with regular block-configurations with blocks of size $2k$. Fig. 11 depicts this generalization: each dot-dashed rectangle with a k inside represents a sequence of k ARs and consequently, two consecutive rectangles form a block of size $2k$. The problem appears when k consecutive ARs fail at the extreme of a block, for example, the failure of the third sequence of k ARs in Fig. 11a. We can solve this issue by modifying the block-configuration in such a way that every connection edge between two neighbors is covered. Fig. 11b shows the recovered system. Notice that the configuration remains unchanged to the right of the bold vertical line whereas it is changed (inverted) to the left. This reconfiguration can be applied the other way around, keeping the left side and reversing the right side. In order to minimize the time to obtain a recovered system in the propagation process, the shortest side should be modified. The new block-configuration can be locally computed with a simple data structure on each agent.

Finally, in case new robots join the team, it is assumed that they enter at the endpoints of the communication line. With this assumption, the ARs can manage the additions easily, making the respective updates in the blocks containing the extremes at which the addition occurs.

5.3. Implementation

In this section we provide pseudo-code of two algorithms running on-board each AR in parallel. One of them addresses the processing of the assembly operations and the other one handles information sharing and reallocations of the assembly operations into the corresponding blocks.

Algorithm 1 On-board algorithm to process the assigned list of assembly parts.

Require: $P^{(i)}$, S
searching ← **false**
carrying ← **false**
p ← **none**
while !ABORT **do**
  if !*sharing* **then**
    if *searching* **then**
      if `found`(p) **then**
        `load`(p)
        searching ← **false**
        carrying ← **true**
        `plan_to_put`(S, p)
      else
        `continue`()
      end if
    else
      if *carrying* **then**
        if `correct_place`(S, p) **then**
          `put`(S, p)
          p ← `next_piece`(S, $P^{(i)}$)
          if p is NOT **none then**
            searching ← **true**
            `search`(p)
          end if
          carrying ← **false**
        else
          `continue`()
        end if
      else
        p ← `next_piece`(S, $P^{(i)}$)
        if p is NOT **none then**
          searching ← **true**
          `search`(p)
        end if
      end if
    end if
  end if
end while
if ABORT **and** *carrying* **then**
  `release`(p)
end if

Algorithm 1 shows the steps to process the assigned list of assembly operations to be performed. Each AR executes the same algorithm with local information. The variables (in italic font), functions and procedures (in math font) used in the algorithm are explained subsequently:

- $P^{(i)}$ is the list of assembly parts assigned to the AR U_i (local task to be performed by U_i).
- S is a description of the whole structure to be built; each part is unique and corresponds to a specific place in S.
- *searching* indicates if the AR is searching for a part.
- *carrying* indicates if the AR is carrying a part with its robotic arm.
- p is the target part to pick or place.
- *sharing* indicates if the robot's communication interface is open (this value is managed by Algorithm 2).

Algorithm 2 Decentralized algorithm to keep a distribution of the assembly parts according to the capabilities of the team members.

Require: T, t, $P^{(i)}$, S, Ids, BS
sharing ← **false**
b ← 0
Q_e ← []
K ← {}
while !ABORT **do**
  if !*sharing* **then**
    if `time_to_share`(T, t) **then**
      c_i ← `current_capabilities`()
      K ← `prepare_info`(K, Pp, $P^{(i)}$, c_i)
      `open_conn`()
      sharing ← **true**
    end if
  else
    if `time_to_close`(T, t) **then**
      `close_conn`()
      B ← `block`(BS, b, Ids)
      P ← `pieces_in_block`(B, K, S)
      C ← `capabilities_in_block`(B, K)
      d ← `division`(P, C)
      $P^{(i)}$ ← $d[i]$
      [Ids, BS] ← `update_team`(Q_e, Ids)
      b ← `next_conf`(b, BS)
      sharing ← **false**
    else
      U ← `meet_neighbor`()
      if `graph_event`(U, Ids) **then**
        e ← `gen_event`(U, Ids)
        Q_e ← $Q_e \cup \{e\}$
      end if
      $info$ ← **none**
      if U is NOT **none then**
        $info$ ← `share`(U, Q_e, K)
      end if
      [Q_e, K] ← `merge_knowledge`($info$, Q_e, K, Ids)
    end if
  end if
end while

- `found`(p) returns **true** if the robot is over the part p.
- `load`(p) activates the robotic arm and picks up part p.
- `plan_to_put`(S, p) elaborates and loads a flight plan to carry part p and place it correctly into structure S.
- `continue`() follows the current flight plan.
- `correct_place`(S, p) returns **true** if the AR has reached the correct position to place part p into S.
- `put`(p) places the part p in its position in S.
- `next_piece`(S, $P^{(i)}$) selects the next piece of $P^{(i)}$ to assemble in S according to the dependency rules and priority of the common parts preventing neighbors from waiting.
- `search`(p) uses the radio signal of p to locate it and elaborates a flight plan to reach it.
- `release`(p) is a protocol to release the part p in case of failure or other cases of mission cancellation.

Algorithm 2 shows the logical steps to share information and reallocate the assembly operations inside the blocks. It uses methods that encapsulate the ideas exposed in the previous section.

- T is the time interval to share information; the AR must share information every T units of time (this value is equal for all the ARs).
- t is the duration of the sharing process, $t < T$.
- Pp is the set of known placed parts.
- Ids is the list of identifiers of all ARs in the team. $Ids[j]$ corresponds to the identifier of U_j and is used to determine if an AR is new in the team or if any AR has failed.
- BS stores a description of the block-configurations to use in the sharing process, including block size and start positions of the blocks. It allows to determine the members composing the block at any given time.
- *sharing* indicates if the communication interface of the robot is open.
- b indicates the current block-configuration in BS.
- Q_e stores a list of detected events in the team: failures and incorporations.
- K stores the knowledge of the ARs in the team (placed and unplaced parts, and current capabilities of the team members).
- `time_to_share`(T, t) returns **true** if it is time to share information.
- `current_capabilities`() estimates its own current capabilities using on-board sensors and time spent performing the last operations.
- `prepare_info`$(K, Pp, P^{(i)}, c_i)$ prepares the local information for sharing (current knowledge, known placed parts, unplaced parts in plan and its own current capabilities).
- `open_conn`() activates and opens the communication interface (to save energy, the communication interface only activated during sharing).
- `time_to_close`(T, t) returns **true** if it is time to close the communication interface.
- `close_conn`() turns the communication interface off if the AR is not sharing information.
- `block`(BS, b, Ids) elaborates a representation of the block to perform a re-allocation of the assembly operations.
- `pieces_in_block`(B, K, S) computes the set of unplaced parts to assign in the block.
- `capabilities_in_block`(B, K) returns the list of the capabilities of the available members in the block.
- `division`(P, C) makes a division between the available members in the block according to their capabilities.
- `update_team`(Q_e, Ids) updates the list of available ARs and the block-configurations according to detected events (failures and incorporations).
- `next_conf`(b, BS) returns the next block-configuration to use (remember that the block-configurations are alternating).
- `meet_neighbor`() connects with a neighbor if possible.
- `graph_event`(U, Ids) detects whether an event has occurred: If the identifier of U is not present in Ids, then U is a new AR incorporated into the team. If U is in Ids but the predecessor or successor is not, then the members between them have failed. If U is **none** then all members from the current position to the bridge end are considered failed.
- `gen_event`(U, Ids) returns a representation of the occurred event to put it in the list of events.
- `share`(U, Q_e, K) sends all the known information to U and returns the information detected by U.
- `merge_info`$(info, Q_e, K, Ids)$ updates the detected information (Q_e and K) using the information received from U.

A video available at https://www.youtube.com/watch?v=BshZ9tSQt9I illustrates the evolution of the task allocation algorithm in a team using these algorithms.

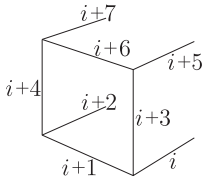

Fig. 12. Enumeration of bridge's parts.

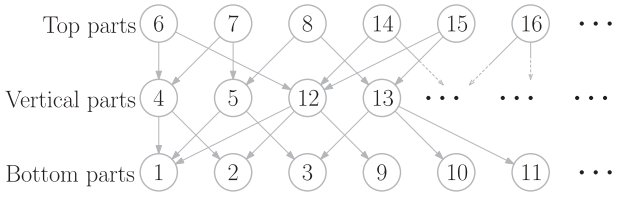

Fig. 13. Precedence graph of the bridge's parts.

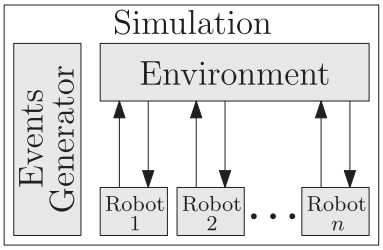

Fig. 14. Simulation architecture.

5.4. Simulations and computational results

In this subsection we provide a detailed description of the implementation of Algorithms 1 and 2 for a simulation of the case study presented above using the BIS strategy. We furthermore analyze the effect of the block size on efficiency and robustness and compare the results with state-of-the-art algorithms.

5.4.1. Parts assignment

In a first step, all parts are ordered so that a subsequence of the sorted list of pieces corresponds to a section of the bridge. The pieces have been numerated following the scheme of Fig. 12 from left to right, see also Fig. 4.

Subsequently, a precedence graph of three levels is constructed. The first level describes bottom parts, the second one vertical parts and the third one top parts. In each level, the pieces are sorted using the assigned index. The precedence graph displaying the numbered parts of the bridge of Fig. 4 is shown in Fig. 13. The idea behind this particular order is to be able to process the pieces in the resulting graph from left to right without violations of the dependency rules. In the example shown in Fig. 13, the ordering is $1, 2, 4, 3, 5, 7, 9, 12, 6, 10, 11, 13, 8, \ldots$. Note that every subsequence of this ordering corresponds to a section of the bridge. Furthermore, in a sharing stage, the set of unplaced pieces is ordered using the positions of the pieces in this total ordering and then is divided according to the capabilities of every robot in the block in order to obtain a workspace division in the new task allocation. Note that through working in this manner, the parts assigned to each robot in the block belong to disjoint sections in the bridge.

5.4.2. Simulation architecture

The architecture of our simulation is shown in Fig. 14: The robots are equipped with an implementation of the previously presented algorithms. The *environment* entity simulates all sensor-related processes of the robots. It manages the positions of the

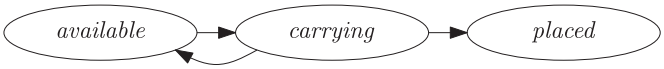

Fig. 15. Considered states of the pieces and the transitions between them.

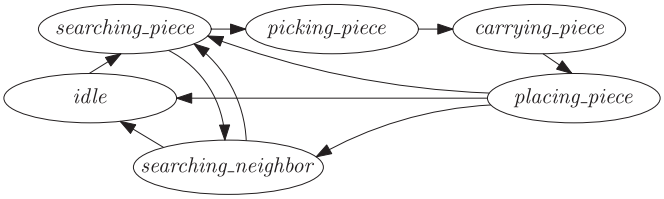

Fig. 16. The states of each robot and the transitions between them.

robots and pieces, the state of the pieces and takes control of simulation rules.

There are three states for pieces in a simulation:

- *available* (the piece is available to be taken by a robot),
- *carrying* (the piece has been loaded by a robot and it is moving towards the structure), and
- *placed* (state reached when the robot carrying the piece has found the correct position of this piece in the structure and placed it).

Fig. 15 shows the transitions between these states. A piece passes from *carrying* state to *available* only when the carrying robot leaves the team.

We define six states for the ARs:

- *idle* (when the simulation starts, or when it finishes the current list of assigned parts),
- *searching_piece* (the robot is searching for a piece to place),
- *picking_piece* (the robot found the searched piece and is picking it up),
- *carrying_piece* (the robot picked up the target piece and is carrying it to the bridge location),
- *placing_piece* (the robot found the final position of the piece and is placing it, after that, the robot decides the next piece to place and passes to the *searching_piece* state) and
- *searching_neighbor* (this state is reached from the states *searching_piece* and *placing_piece* when the time frame to meet a neighbor has passed).

Fig. 16 shows the transitions between robot states: If a robot is in *placing_piece* state and has completed the list of assigned assembly operations, it passes to *idle* state. When a robot is in *searching_neighbor* state and meets a neighbor or reaches the bridge end, it proceeds as follows: if there are assembly operations to perform, it determines the next piece to place and passes to *searching_piece* state, but if there are no assembly operations to perform (in its assigned list), it passes to *idle* state.

In our simulation, we introduce the *skill coefficient* of a robot as a value to model its ability to manipulate pieces. Higher values of ability correspond to lower required time to lift or place a piece. A robot with a robotic arm with 6 degrees of freedom has a higher skill coefficient than a robot with 4 degrees of freedom. Also, this coefficient can be interpreted as the operability state of the robotic arm (a robotic arm with mechanical problems has a skill coefficient lower than a robot in perfect working conditions).

The *environment* manages the transitions between the states. For instance, a robot can only lift up a piece if the piece is available and the robot is flying over the piece. It furthermore controls the advance and current state of the lifting and placing operations according to the current skill coefficient of the robot, and keeps track of the positions of the robots according to their motion directions and speeds. Note, that in this case the capability coefficient of a robot is a combination of speed and skill.

The other important entity in our simulation is the *events generator*, which generates random events of mission abandonment, incorporation of members and changes on the capabilities (speed or skill coefficient) of a specific robot.

5.4.3. Scenario description

We consider for the simulation a scenario where the bridge to be built is composed of bars with a length of one meter. All pieces have been dropped in a single place (store) at a distance of 20 meters from the bridge location. The communication range of the robots is 8.0 meters, their initial speed ranges from 3.7 to 4.0 meters per second and their initial skill coefficients are set between 2.9 and 3.0 (randomly distributed following a uniform distribution). We are assuming that the needed effort to lift and assemble a piece is 18 and 90, respectively. Thus, a robot with skill coefficient 3.0 spends $18/3.0 = 6$ seconds to lift a piece and $90/3.0 = 30$ seconds to place it in the structure. Note that the measure of the effort to place a single piece correctly is a combination of the effort to lift it, the required effort to transport it to the final location in the structure and the effort to place it.

Fig. 17 shows the different states of a team of eight robots during a simulation. At the beginning, robots are located above the structure location (Fig. 17a) and have an initial assembly assignment. After assessing the first required piece, the robots head towards the store (Fig. 17b and c). Note that various robots can be at the store simultaneously. In our simulation, the store is represented as a single point and we assume that the robots have implemented a local strategy of collision avoidance. In real-word applications the store is usually a region.

Fig. 17d and e show the robots on their way to the structure location carrying the pieces to be placed. Note that the robots in the middle reach the structure first and the robots closest to the bridge ends last. In Fig. 17e it can be seen that, while robot 3 is assembling a piece, robot 0 is still carrying a piece to the correct place in the structure. In this simulation, the sharing process is not marked by time intervals since a robot executes the local sharing process when it has new information of all the members in the current block. To avoid endless waits, a timeout for information sharing takes place when a member in the block fails. Fig. 17f shows an arbitrary instant illustrating how robots move independently from the store to the structure, execute their assembly operations and exchange information when connected with a neighbor near the structure (the pair of robots (0, 1) and (6, 7) may be sharing information). The video https://youtu.be/0m6J1cqO1PI shows the initial stages of a simulation.

5.4.4. Results with different strategies

Two possible information sharing procedures can be applied to the scenario applied above, a local or a global information retrieval before the decision process. So far, related approaches, the *one-to-one* and the *centralized* strategy, represent strict versions of these choices. We compare such algorithms to the BIS strategy introduced in this paper. In the one-to-one strategy, the information is shared between only two neighboring robots. This approach has been used frequently in related literature (Acevedo et al., 2013a; Acevedo, Arrue, Maza, & Ollero, 2013b; Acevedo et al., 2014). On the other end of the spectrum are centralized algorithm with global knowledge, where a new allocation is created based on all the information available in the entire system. However, the BIS strategy is decentralized and takes advantage of quick local decisions using a group of agents to achieve global solutions. Note that the block size in the BIS strategy indicates the degree of global knowledge in the reallocation process. Furthermore the three algorithms can be integrated within the BIS strategy, namely, size 2 for

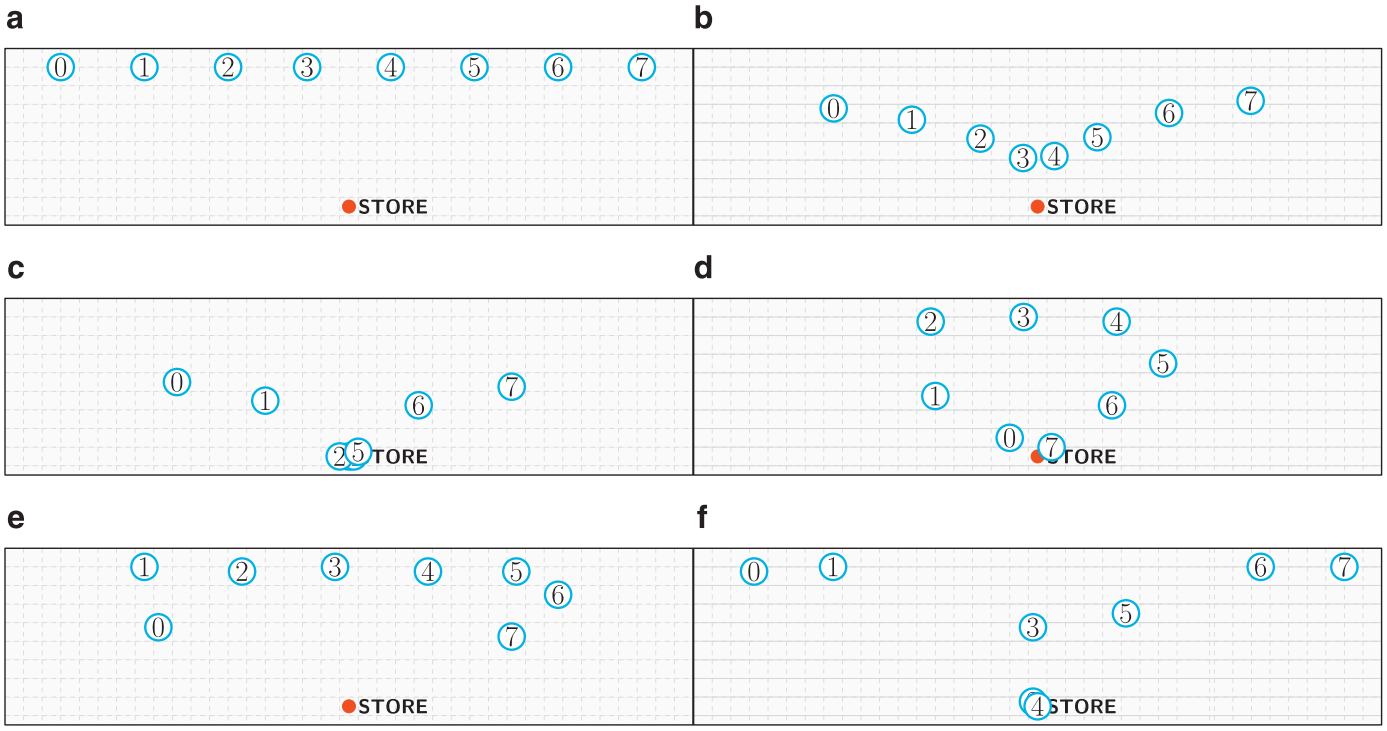

Fig. 17. Motion of the robots between the structure and the store.

one-to-one and size n for the centralized approach, where n is the number of robots in the team.

Using blocks of big size implies a more centralized approach which allows the computation of a near optimal solution since global knowledge is taken into account. However, collecting, processing, and broadcasting all globally available knowledge can be time consuming which limits the efficiency of the overall system. Additionally, an almost centralized approach may not yield improvement: If some agents in a large block leave the team, the task allocation cannot be completed until the failure has been detected by all other block members. In contrast, a more decentralized approach using small blocks achieves robustness to individual failures and may gain a performance advantage through parallel computation. However, the quality of the solution may decrease. In this section we explore this trade-off between efficiency and robustness.

In order to analyze the balance of the task allocation in the team during a mission we introduce some notations. Let $P_i(t)$ and $c_i(t)$ be the number of unplaced pieces (workload) assigned to the ith robot and its capability coefficient at instant t of the mission, respectively. From Eq. (2) it follows that in a team with n robots at instant t, the task allocation is balanced if:

$$\frac{P_j(t)}{c_j(t)} = \frac{\sum_{i=1}^{n} P_i(t)}{\sum_{i=1}^{n} c_i(t)}, \quad \text{for all } 1 \le j \le n. \tag{8}$$

Let $r_i(t) = P_i(t)/c_i(t)$ be the ratio of workload and capability of the ith robot at instant t and let $r^*(t) = \sum_{i=1}^{n} P_i(t) / \sum_{i=1}^{n} c_i(t)$ be the optimum ratio of workload and capability at time instant t according to (8).

In the scope of this study, we consider the following function to assess the imbalance of the workload in the system:

$$\xi(t) = \sqrt{\frac{\sum_{i=1}^{n} (r_i(t) - r^*(t))^2}{n}}. \tag{9}$$

Note that if the workload in the team is balanced then the value of ξ is zero, while a higher value of ξ corresponds to a greater degree of imbalance.

Fig. 18 depicts the comparison among different strategies for a setup with 8 robots and a bridge of 100 cubes length. The tested

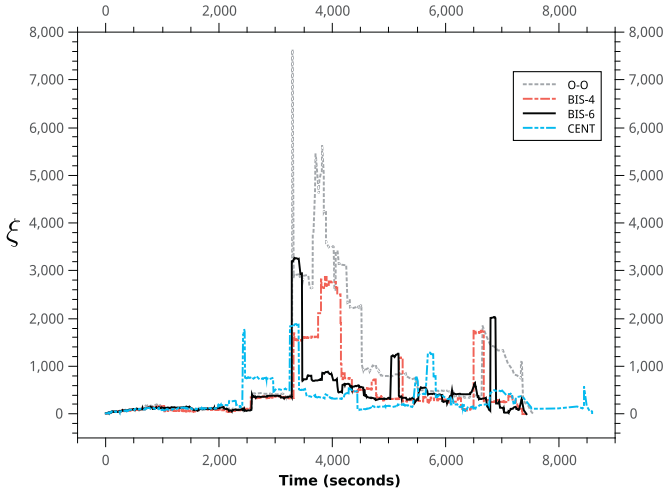

Fig. 18. Behavior of the function ξ in a simulation using different strategies: the one-to-one strategy (O-O, dashed gray line); BIS strategy with block size 4 (BIS-4, dash-dotted red line); BIS strategy with block size 6 (BIS-6, continuous black line) and centralized strategy (CENT, dash-dot-dotted blue line). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

strategies are: one-to-one, centralized and BIS strategy. The latter was applied in two regular block-configurations with blocks of size 4 and 6, respectively. We will denote by *O-O*, *CENT* and *BIS-k* the strategies one-to-one, centralized and BIS strategy with block size k, respectively. The results indicate that CENT is the most time consuming. The team spends 7554.5 seconds with O-O, 7446.5 seconds with BIS-4, 7433.5 seconds with BIS-6 and 8594.0 seconds with CENT. During the simulation, the *events generator* produces two random fail-events (consequently information sharing is needed to finish the construction of the bridge).

In Fig. 18, every peak in a curve represents a disturbance in the workload balance due to a sharing process. Initially, the team starts with an optimal task allocation and thus, at the beginning of the

Table 1
Summary of *end time*, *mean* value of function ξ and *maximum* value of function ξ using different strategies for a setup with 10 robots and different bridge lengths. The average, best and worst value of each of these parameters is shown. The tables (a), (b), (c) and (d) corresponds to bridges of 40, 60, 80 and 100 cubes length, respectively.

	End Time (seconds)			mean-ξ			max-ξ		
	Avg	Best	Worst	Avg	Best	Worst	Avg	Best	Worst
O-O	**1389.53**	1325.00	1636.00	50.59	29.94	237.32	308.73	56.18	2666.22
BIS-4	1434.30	1311.00	2248.00	**47.39**	38.91	67.14	309.49	65.28	3008.07
BIS-6	1467.51	1315.50	1978.00	52.74	41.49	61.92	199.44	79.32	2827.56
BIS-8	1514.89	1322.50	3344.50	57.36	27.35	71.07	**123.82**	89.77	853.34
CENT	**2031.71**	1348.00	6587.50	**64.84**	43.11	86.07	**856.64**	94.70	2826.23

(a)

	End Time (seconds)			*mean-ξ*			*max-ξ*		
	Avg	*Best*	*Worst*	*Avg*	*Best*	*Worst*	*Avg*	*Best*	*Worst*
O-O	**2494.09**	2222.50	3243.00	**198.21**	42.81	866.00	2344.04	72.68	4554.40
BIS-4	2523.77	2189.50	3351.50	**87.18**	46.38	226.73	2162.54	88.97	4180.97
BIS-6	2658.44	2222.00	4675.00	99.58	63.37	304.14	2176.64	114.84	5140.51
BIS-8	2694.95	2203.00	3937.00	88.02	56.37	153.99	**2114.48**	127.39	4647.83
CENT	**3525.19**	2256.00	4722.50	130.69	47.08	411.49	**2904.37**	105.25	4979.16

(b)

	End Time (seconds)			*mean-ξ*			*max-ξ*		
	Avg	*Best*	*Worst*	*Avg*	*Best*	*Worst*	*Avg*	*Best*	*Worst*
O-O	4243.14	3523.50	6783.00	**506.05**	66.79	1376.72	**5035.26**	347.14	9060.06
BIS-4	4326.19	3469.00	6131.50	268.84	59.96	806.60	4994.50	112.68	7387.02
BIS-6	**4232.80**	3496.00	7645.50	206.53	77.42	595.44	4987.09	131.83	7424.13
BIS-8	4311.82	3501.00	6269.50	**190.56**	91.45	603.06	**4854.98**	169.49	7510.42
CENT	**4989.96**	3479.00	7415.5	331.48	62.51	827.66	4886.42	119.46	6950.91

(c)

	End Time (seconds)			*mean-ξ*			*max-ξ*		
	Avg	*Best*	*Worst*	*Avg*	*Best*	*Worst*	*Avg*	*Best*	*Worst*
O-O	6500.33	5656.50	8622.00	**1078.28**	112.09	2518.39	**6831.81**	205.28	10282.73
BIS-4	**6392.32**	5494.00	8276.50	490.07	80.61	1181.09	6717.91	142.26	10380.09
BIS-6	6509.25	5564.00	8850.50	383.30	101.29	864.76	**6546.49**	174.50	9953.94
BIS-8	6436.99	5532.00	8896.00	**336.21**	119.54	876.99	6787.17	452.30	8624.41
CENT	**6719.93**	5740.00	8982.50	476.96	130.54	757.68	6618.10	217.68	9027.23

(d)

simulation, the values of ξ are close to 0. Around second 2400 of the simulation, we observe an abrupt increase of imbalance for the team using the centralized strategy. The cause of this behavior is the high volume of information which needs to be collected. The members of the team spend a relatively large amount of time retrieving information from all teammates. In fact, when the leftmost robot receives the information of the rightmost robot, the information is already outdated. Suppose the identifiers of the robots from left to right are $r_1, r_2, \ldots, r_8$ and robot r_8 shares information with its neighbor r_7. At this moment, r_7 receives the information about the number of unplaced pieces assigned to r_8. At a certain instant in time, r_7 passes this information to r_6, and so on until this information reaches r_1. While the information about unplaced pieces assigned to r_8 is traveling to reach r_1, r_8 continues working. Therefore, at the moment in which r_1 receives the information about r_8, the number of unplaced pieces assigned to r_8 that receives r_1 is greater than the actual number of unplaced pieces assigned to r_8. This causes an alteration in the task allocation, assigning some placed pieces to the robots as unplaced pieces.

Around instant 3250 seconds of the simulation, an event occurs which produces an imbalanced workload (note that all the curves have a strong disturbance). The curve corresponding to the one-to-one strategy experiences stronger disruption than the others due to the following reason: When a robot detects that its neighbor has failed, it resumes the assembly task of its neighbor and its workload grows significantly with respect to other workloads. However, using blocks of robots (with sizes greater than 2) in the reallocation, the assembly task is divided between the remaining members of the block thus obtaining better distribution (this argument also applies to the centralized strategy).

Larger blocks imply fewer reallocation operations in order to obtain an optimal task allocation. Therefore, in most cases faster convergence to an optimal partition is guaranteed. Note in Fig. 18 that the team using one-to-one spends more time to reach a balanced task allocation from a peak than others. On the other hand, if the changes in the system occur faster than the broadcast time in the block, the convergence to an optimal task allocation may be put on hold. As an example, see the behavior shown in Fig. 18 around second 2400 of the simulation using the centralized strategy. Also, using the BIS strategy with larger blocks (or a centralized strategy) may cause larger time intervals between reallocations. This may result in a problem if for instance a robot u is trying to place a piece which requires another piece assigned to a neighbor u', and u' fails. Then u will be locked trying to place its piece until the next reallocation occurs. Another undesirable situation occurs if a robot u is idle (with no assigned pieces because it has finished its assigned assembly operations) while its neighbor u' still has pending operations. In this case, u will be idle and waiting for a new assignment until the next reallocation process occurs. Both situations may increase the execution time of the global task as illustrated in Fig. 18 by the curve corresponding to the centralized strategy.

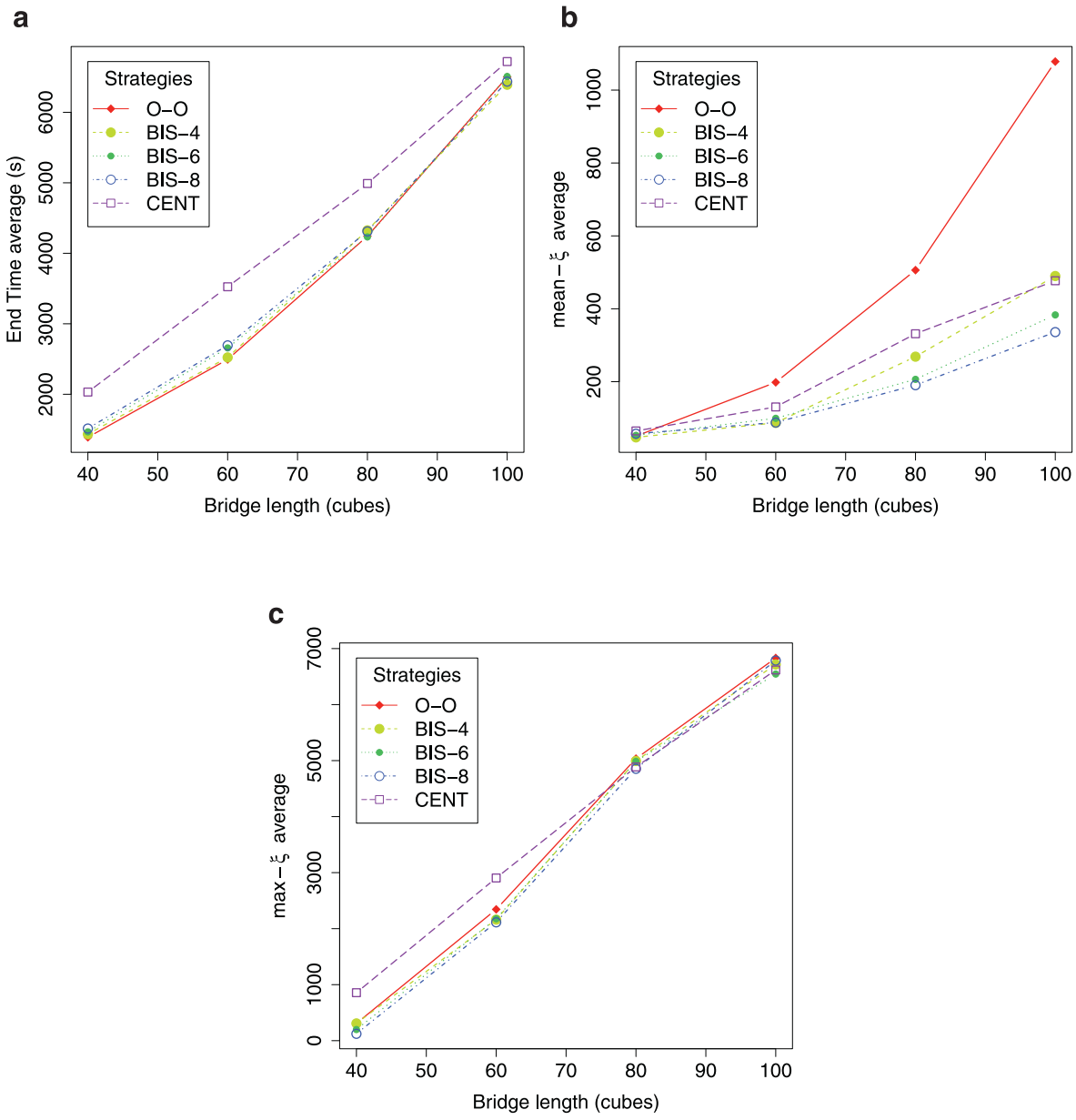

Fig. 19. Performance of End Time (a), mean-ξ (b) and max-ξ (c).

In summary, in this specific experiment, BIS-6 seems to be the most adequate choice to maintain a balance between efficiency and robustness.

Finally, in order to analyze both, workload variability and total time needed to assemble the structure, for the chosen strategy, we performed several experiments using a team starting with 10 robots. Different bridge lengths were tested: 40, 60, 80 and 100 cubes. For each length, we performed 50 runs using different strategies: O-O, BIS-4, BIS-6, BIS-8 and CENT. We have implemented the scenario introduced in Section 5.4.3 and the occurrence of failures in the system has been generated randomly.

Table 1a –d show the obtained results for each bridge length. We have considered three parameters: the total time to assemble the bridge (*End Time*) in seconds, the mean value of function ξ (*mean-*ξ) and the maximum value of function ξ (*max-*ξ). The tables show the average, best and worst cases (*Avg*, *Best* and *Worst*, respectively) for each parameter. The best and worst average value of these parameters have been highlighted in the tables. Fig. 19 illustrates the growth of the parameters using the average data.

The results (Table 1a–d) indicate that the centralized strategy is the worst among the considered methods: it obtained the lowest efficiency and furthermore ranked low with respect to balancing. The other strategies appear to have a similar behavior regarding efficiency, whereas BIS shows a better performance with respect to balancing compared to O-O.

6. Conclusions and future developments

In this paper we have introduced the block-information-sharing strategy as a new paradigm for performing task allocation in a decentralized manner. This strategy is applicable to a variety of scenarios meeting the following conditions: the general task should be divisible and parallelizable, the communication graph should be connected and the union of the selected block-configurations should be a covering of the set of edges of the communication

graph. It has been proved that if these conditions are fulfilled, the convergence to an optimal task allocation can be guaranteed. Thus, the experimental proofs on the convergence in related literature (Acevedo et al., 2013a; Acevedo et al., 2013b; Acevedo et al., 2014; Caraballo et al., 2014) are now theoretically endorsed.

It has also been demonstrated that the block size is an important parameter in this strategy which is directly related to the rate of convergence and robustness. In an ideal scenario with instantaneous communication and no limit in the information amount to be handled by one robot, the convergence rate increases with increasing block size. However, in real applications, sharing with blocks of big size may be expensive and the system can lose in efficiency and fault tolerance. In fact, the memory of the robots is limited and time is spent while moving to ensure the information sharing process. Thus, although the benefits of the BIS strategy are visible in the computational experiments in both, surveillance tasks (Caraballo et al., 2014) and assembling structures, the block size is a parameter that can be adequately set based on prior studies and simulations for a given scenario. Indeed, the lack of a priori knowledge of the block size for a specific communication graph is the bottleneck of the BIS paradigm. In this work, the block size is considered an input parameter, whereas the computation of an optimal value for a given configuration remains an open problem.

References

Acevedo, J., Arrue, B., Díaz-Báñez, J., Ventura, I., Maza, I., & Ollero, A. (2013a). Decentralized strategy to ensure information propagation in area monitoring missions with a team of UAVs under limited communications. In *Proceedings of the international conference on unmanned aircraft systems (ICUAS 2013)* (pp. 565–574).

Acevedo, J., Arrue, B., Maza, I., & Ollero, A. (2013b). Distributed approach for coverage and patrolling missions with a team of heterogeneous aerial robots under communication constraints. *International Journal of Advanced Robotic Systems, 10*(28), 1–13.

Acevedo, J., Arrue, B. C., Díaz-Báñez, J., Ventura, I., Maza, I., & Ollero, A. (2014). One–to-one coordination algorithm for decentralized area partition in surveillance missions with a team of aerial robots. *Journal of Intelligent and Robotic Systems, 74*(1–2), 269–285.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183*(2), 674–693.

Cao, Y. U., Fukunaga, A. S., & Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots, 4*(1), 7–27.

Caraballo, L., Acevedo, J., Díaz-Báñez, J., Arrue, B., Maza, I., & Ollero, A. (2014). The block-sharing strategy for area monitoring missions using a decentralized multi-uav system. In *Proceedings of the 2014 international conference on unmanned aircraft systems* (pp. 602–610). IEEE.

Dias, M. B. (2004). *Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments* Ph.D. thesis.. Carnegie Mellon University Pittsburgh.

Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE, 94*(7), 1257–1270. doi:10.1109/JPROC.2006.876939.

Gerkey, B., & Matarić, M. (2002). Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation, 18*(5), 758–768.

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

Ghandi, S., & Masehian, E. (2015). Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design, 67–68*, 58–86.

Jimenez, P. (2011). Survey on assembly sequencing: a combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing*, 1–16.

Kavraki, L., Latombe, J.-C., & Wilson, R. H. (1993). On the complexity of assembly partitioning. *Information Processing Letters, 48*(5), 229–235.

Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks* (pp. 31–51). Springer International Publishing.

Korsah, G. A., Stentz, A., & Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research, 32*(12), 1495–1512.

Lemaire, T., Alami, R., & Lacroix, S. (2004). A distributed tasks allocation scheme in multi-uav context. In *Proceedings of 2004 IEEE international conference on robotics and automation, 2004: vol. 4* (pp. 3622–3627).

Rashid, M., Hutabarat, W., & Tiwari, A. (2012). A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *The International Journal of Advanced Manufacturing Technology, 59*(1–4), 335–349.

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*(3), 666–693. Balancing Assembly and Transfer lines.

Sempere, A., Llorente, D., Maza, I., & Ollero, A. (2014). Local heuristics analysis in the automatic computation of assembly sequences for building structures with multiple aerial robots. In *Proceedings of ROBOT 2013: first Iberian robotics conference: vol. 252* (pp. 87–101). Springer International Publishing.

Shima, T., Rasmussen, S. J., Sparks, A. G., & Passino, K. M. (2006). Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. *Computers and Operations Research, 33*(11), 3252–3269. Part Special Issue: Operations Research and Data Mining.

Tsalatsanis, A., Yalcin, A., & Valavanis, K. P. (2012). Dynamic task allocation in cooperative robot teams. *Robotica, 30*, 721–730.