Autonomous Robots (2018) 42:1477–1495
https://doi.org/10.1007/s10514-018-9742-5

Solving the task variant allocation problem in distributed robotics

José Cano[1] · David R. White[3] · Alejandro Bordallo[1] · Ciaran McCreesh[2] · Anna Lito Michala[2] · Jeremy Singer[2] · Vijay Nagarajan[1]

Received: 1 March 2017 / Accepted: 2 April 2018 / Published online: 25 April 2018

Abstract
We consider the problem of assigning software processes (or tasks) to hardware processors in distributed robotics environments. We introduce the notion of a *task variant*, which supports the adaptation of software to specific hardware configurations. Task variants facilitate the trade-off of functional quality versus the requisite capacity and type of target execution processors. We formalise the problem of assigning task variants to processors as a mathematical model that incorporates typical constraints found in robotics applications; the model is a constrained form of a multi-objective, multi-dimensional, multiple-choice knapsack problem. We propose and evaluate three different solution methods to the problem: constraint programming, a constructive greedy heuristic and a local search metaheuristic. Furthermore, we demonstrate the use of task variants in a real instance of a distributed interactive multi-agent navigation system, showing that our best solution method (constraint programming) improves the system's quality of service, as compared to the local search metaheuristic, the greedy heuristic and a randomised solution, by an average of 16, 31 and 56% respectively.

Keywords Task allocation · Distributed robotics · Multi-robot systems · Multi-objective optimisation

1 Introduction

Modern robotics systems are increasingly distributed, heterogeneous and collaborative, incorporating multiple independent agents that communicate via message passing and distributed protocols. A distributed approach can offer desirable qualities such as improved performance. Heterogeneity refers to the type and amount of hardware resources (e.g. sensors, CPU capacity) available on each agent in the system. In such systems, the efficient allocation of software processes (referred to as *tasks*) to hardware processors is of paramount importance in ensuring optimality. Previous works (Lee et al. 2014; Liu and Shell 2012) generally take an approach that considers only a fixed set of tasks, equivalent to a "one size fits all" architecture, limiting the ability of a system to adapt to different hardware configurations, and reducing the opportunities for optimisation.

Instead, we advocate the development of systems based on the selection and allocation of what we term "task variants". Task variants are interchangeable software components that offer configurable levels of quality of service (QoS) with a corresponding difference in the amount and/or type of computing resources they demand; such variants naturally arise in many scenarios, and often deployed systems consist of a particular subset of variants that have been implicitly chosen by a system architect. For example, consider alternative feature detection algorithms to solve a common task in a robotics vision pipeline: different algorithms provide increasingly sophisticated recognition methods but at the cost of increasing CPU load. Similarly, a variant may offer accelerated processing by targeting specialised hardware (e.g. GPUs, FPGAs).

Currently, the crucial step of selecting and allocating such task variants is typically performed using ad-hoc methods, which provide no guarantee of optimality and may thus lead to inefficient allocation. In this paper, we take a more systematic approach. We formalise the task variant allocation

This is one of several papers published in *Autonomous Robots* comprising the "Special Issue on Robotics Science and Systems".

✉ José Cano
jcanore@inf.ed.ac.uk

1 School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK

2 School of Computing Science, University of Glasgow, Glasgow G12 8RZ, UK

3 Department of Computer Science, University College London, London WC1E 6BT, UK

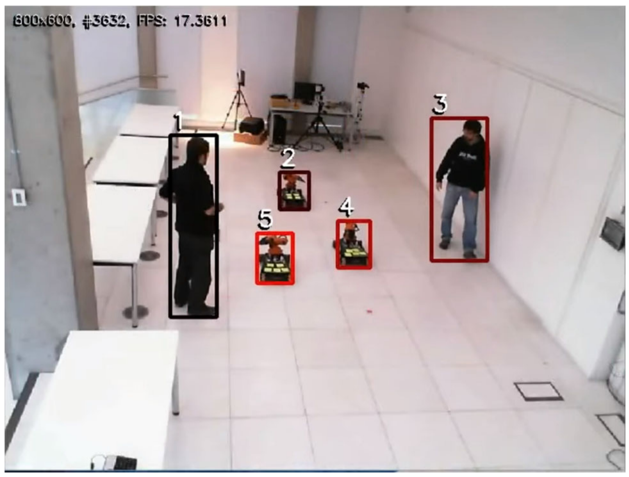


Fig. 1 Case study: multi-agent navigation system composed of autonomous robots (KUKA youBots), humans, and network cameras

problem and propose three different solution methods that are able to efficiently exploit the available resources with the objective of maximising QoS while ensuring system correctness.

We focus on distributed heterogeneous robotics systems where variants are naturally available for several tasks. In particular, our work has been driven by a case study (Fig. 1), in the form of a distributed system of agents running on ROS (Quigley et al. 2009). The application implements a framework for inferring and planning with respect to the movement of goal-oriented agents in an interactive multi-agent setup—full details can be found in Bordallo et al. (2015). There are two types of agents navigating in the same physical space: autonomous robots represented by KUKA youBots (Bischoff et al. 2011) and humans. Each agent is pursuing a goal (a specific spatial position in the scenario) while avoiding collisions with other agents, based on online sensor processing and beliefs concerning the latent goals of other agents.

Specific tasks are used to accomplish this objective in a distributed fashion. For example, robots infer navigation goals of other agents from network camera feeds, provided by at least one *Tracker* task—meanwhile humans act independently and are assumed to navigate as a rational goal-oriented agent through the space. Some tasks can be configured via parameter values (e.g. the camera frame rate for the *Tracker* task) that translate into variants for that task. Each of these variants produces a different level of QoS, which we assume is quantified by an expert system user. Thus, the objective is to select task variants and allocate them to processors so as to maximise the overall QoS while agents reach their goals.

The contributions of the paper are as follows: (i) we introduce a mathematical model that represents the task variant selection and allocation problem; (ii) we propose three different solution methods (constraint programming, local search metaheuristic, greedy heuristic) to the problem; (iii) we evaluate and compare the solution methods through simulation; (iv) we validate the solution methods in a real-world interactive multi-agent navigation system, showing how our best solution method (constraint programming) clearly outperforms the average QoS of the local search metaheuristic by 16%, the greedy heuristic by 31%, and a random allocation by 56%. To the best of our knowledge, we are the first to address task allocation in the presence of variants in distributed robotics.

2 Problem formulation

We now model the problem of task variant allocation in distributed robotics, in a general formulation that also applies to the specifics of our case study. We consider allocation as a constrained type of multi-objective, multi-dimensional, multiple-choice knapsack problem. Whilst instances of these three problems are individually common in the literature (Kellerer et al. 2004; Martello and Toth 1990), the combination is not. In addition, we allow for a number of unusual constraints describing task variants that distinguish this formulation from previous work (e.g. the specific type of hardware required to run a variant). Note that in this work we consider task allocation from a static point of view, although a dynamic case could be addressed as a process of repeated static allocations, or a more sophisticated method could be developed. We leave the dynamic case for future work.

Our formulation of the problem divides cleanly into three parts: the *software* architecture of the system, including information about task variants; the *hardware* configuration that is being targeted as a deployment platform; and the constraints and goals of task *selection and allocation*, which may be augmented by a system architect.

2.1 Software model

A software architecture is defined by a directed graph of tasks, (T, M) where the set of tasks $T = \{\tau_1 \dots \tau_n\}$ and each task τ_i is a unit of abstract functionality that must be performed by the system. Tasks communicate through message-passing: edges $m_{i,j} = (\tau_i, \tau_j) \in M \subseteq T \times T$ are weighted by the 'size' of the corresponding message type, defined by a function $S : m_{i,j} \to \mathbb{N}$; this is an abstract measure of the bandwidth required between two tasks to communicate effectively.

Tasks are fulfilled by one or more *task variants*. Each task must have at least one variant. Different variants of the same task reflect different trade-offs between resource requirements and the QoS provided. Thus a task τ_i is denoted as set of indexed variants: $\tau_i = \{v_1^i \dots v_n^i\}$, $\tau_i \neq \emptyset$. For convenience, we define $V = \cup_i \tau_i$, such that V is the set of all variants across all tasks. For simplicity, we make the conservative assumption that the maximum message size for a task τ_i is the same across all variants v^i_j of that task, and we use this maximum value when calculating bandwidth usage for any task variant. Note that this assumption could only impact the overall solution in highly constrained networks, which is very unlikely nowadays. For example, in our case study the maximum data rates required are very low (45 KB/s) compared to the available bandwidth (300 MB/s).

A given task variant v^i_j is characterised by its processor utilisation and the QoS it provides, represented by the functions $U, Q : v^i_j \rightarrow \mathbb{N}$. The utilisation of all task variants is expressed normalised to a 'standard' processor; the capacity of all processors is similarly expressed. QoS values can be manually (Sect. 5.1) or automatically generated (future work), although this is orthogonal to the problem addressed.

2.2 Hardware model

The deployment hardware for a specific system is modelled as an undirected graph of processors, (P, L) where the set of processors $P = \{p_1 \ldots p_n\}$ and each processor p_k has a given processing capacity defined by a function $D : p_k \rightarrow \mathbb{N}$. A bidirectional network link between two processors p_k and p_m is defined as $l_{k,m} = (p_k, p_m) \in L \subseteq P \times P$, so that each link between processors will support one or more message-passing edges between tasks. The capacity of a link is given by its maximum bandwidth and is defined by a function $B : l_{k,m} \rightarrow \mathbb{N}$. If in a particular system instance multiple processors share a single network link, we rely on the system architect responsible for specifying the problem to partition network resources between processors, such as simply dividing it equally between processor pairs.

2.3 Selection and allocation problem

The problem hence is to find a partial function $A : V \rightarrow P$, that is, an assignment of task variants to processors that satisfies the system constraints (i.e. a *feasible* solution), whilst maximising the QoS across all tasks, and also maximising efficiency (i.e. minimising the average processor utilisation) across all processors. As A is a partial function, we must check for domain membership of each task variant, represented as $dom(A)$, to determine which variants are allocated.

We assume that if a processor is not overloaded then each task running on the processor is able to complete its function in a timely manner, hence we defer the detailed scheduling policy to the designer of a particular system.

An optimal allocation of task variants, A^*, must maximise the average QoS across all tasks (i.e. the global QoS):

$$max \quad 1/|T| \sum_{v^i_j \in dom(A)} Q(v^i_j) \tag{1}$$

Whilst minimising the average utilisation across all processors *as a secondary goal*:

$$min 1/|P| \sum_{p_k \in P} \sum_{v^i_j \in dom(A):\ A(v^i_j)=p_k} U(v^i_j) \tag{2}$$

Exactly one variant of each task must be allocated:

$$\forall \tau_i \in T,\ \forall v^i_j, v^i_k \in \tau_i :$$
$$(v^i_j \in dom(A) \wedge v^i_k \in dom(A)) \implies j = k \tag{3}$$

The capacity of any processor must not be exceeded:

$$\forall p_k \in P : \Big(\sum_{v^i_j \in dom(A):\ A(v^i_j)=p_k} U(v^i_j) \Big) \leq D(p_k) \tag{4}$$

The bandwidth of any network link must not be exceeded:

$$\forall l_{q,r} \in L : \Big(\sum_{i:A(v^i_j)=p_q} \sum_{k:A(v^k_l)=p_r} S(m_{i,k}) + S(m_{k,i}) \Big) \leq B(l_{q,r}) \tag{5}$$

In addition, *residence constraints* restrict the particular processors to which a given task variant v^i_j may be allocated, to a subset $R^i_j \subseteq P$. This is desirable, for example, when requisite sensors are located on a given robot, or because specialised hardware such as a GPU is used by the variant:

$$v^i_j \in dom(A) \implies A(v^i_j) = p_k \in R^i_j \tag{6}$$

Coresidence constraints limit any assignment such that the selected variants for two given tasks must always reside on the same processor. In practice, this may be because the latency of a network connection is not tolerable. The set of coresidence constraints is a set of pairs (τ_i, τ_k) for which:

$$\forall v^i_j \in \tau_i, \forall v^k_l \in \tau_k : (v^i_j \in dom(A) \wedge v^k_l \in dom(A))$$
$$\implies A(v^i_j) = A(v^k_l) \tag{7}$$

3 Solution methods

We now propose and describe our three different centralised approaches to solving the problem of task variant allocation[1]: constraint programming (CP), a greedy heuristic (GH), and local search metaheuristic (LS). These are three broadly representative search techniques from diverse families of solution methods, as outlined by Gulwani (2010).

[1] All the source code can be found online (White and Cano 2017).

3.1 Constraint programming

We expressed the problem in MiniZinc 2.0.11 (Nethercote et al. 2007), a declarative optimisation modelling language for constraint programming. A MiniZinc model is described in terms of variables, constraints, and an objective. Our model has a variable for each variant, stating the processor it is to be assigned to; since we are constructing a partial mapping, we add a special processor to signify an unassigned variant. Matrices are used to represent the bandwidth of the network and the sizes of messages exchanged between tasks.

Most constraints are a direct translation of those in Sect. 2.3 although the constraint given by Eq. 3 is expressed by saying that the sum of the variants allocated to any given task is one—this natural mapping is why we selected MiniZinc, rather than (for example) encoding to mixed integer programming (Wolsey 2008). The development of a model that allows MiniZinc to search efficiently is key to its success, and we spent some time refining our approach to reduce solution time. However, the model could be further refined. For example, we could investigate non-default variable and value ordering heuristics, or introduce a custom propagator which avoids the $O(n^4)$ space associated with encoding the bandwidth constraints.

There are two objectives to be optimised, and we achieve this by implementing a two-pass method: first the QoS objective is maximised, we parse the results, and then MiniZinc is re-executed after encoding the found optimal value as a hard constraint whilst attempting to minimise processor utilisation—note that the MiniZinc model doesn't include the $1/|P|$ and $1/|T|$ terms in the objective, since floats or divisions affect constraint programming performance considerably. Instead, we apply these terms in the Python program that calls MiniZinc.

The full model is available online (White and Cano 2017), but to give a flavour, we show our variables, a constraint, and the first objective:

```
array[1..nVariants] of
  var 0..nProcessors: assignments;

constraint forall (p in 1..nProcessors) (
    sum([if assignments[v] == p
         then utilisations[v]
         else 0
         endif
        | v in 1..nVariants])
      <= capacities[p]);

solve maximize sum(
  [if assignments[v] != 0
   then qos[v]
   else 0
   endif
   | v in 1..nVariants]);
```

MiniZinc allows instance data to be separated from the model. Part of a data file looks like this:

```
nProcessors = 3;
capacities  = [ 100, 100, 223 ];
links       = [|    -1, 17745, 17676
              | 17745,    -1, 17929
              | 17676, 17929,    -1 |];
```

To solve instances, we used the *Gecode* (Gecode Team 2006) constraint programming toolkit, which combines backtracking search with specialised inference algorithms. We used the default search rules, and only employ standard toolkit constraints. It addition to being used as an exact solver, *Gecode* can also run in *anytime* fashion, such that it reports the best solution found so far. Our system reports its progress in terms of the best-known solution at any point during the execution of the solver, as well as the optimal result, where found. In our evaluation we consider both the standard mode, which returns the global optimum after an unrestricted runtime (Sect. 5.3), and also this anytime mode that returns the best result found so far (Sect. 5.5).

3.2 Greedy heuristic

Our second solution method is a non-exact greedy algorithm that uses a heuristic developed from an algorithm originally designed for solving a much simpler allocation problem (Cano et al. 2015). The procedure is described in Algorithm 1, and attempts to obey constraints, then allocate the most CPU intensive tasks possible to those processors with the greatest capacity.

First, the smallest task variants with residency constraints are allocated to processors (lines 3–7), beginning with the largest capacity processor if the subset R^i_j for a given task variant v^i_j contains more than one element. Next, the smallest variants of any tasks with coresidency constraints are assigned selecting processors from P_{max} (lines 8–10). Then, the smallest variants of any remaining, unallocated, tasks are allocated, again preferring processors with more capacity (lines 11–13). Finally, the algorithm iteratively attempts to substitute smaller variants with larger ones on the same processor (lines 14–17). Note that the way in which the next processor (from R^i_j, P_{max}) or variant is selected must also ensure that allocations will not result in a violation of any previously satisfied constraints.

The greedy heuristic is not guaranteed to find a solution, but if it finds one it is always feasible, i.e. satisfies the system constraints. The ability to provide solutions is greatly determined by residency and coresidency constraints.

Algorithm 1 Greedy Heuristic

```
1: $P_{max}$ = sort $processors$ by max capacity
2: $T_{max}$ = sort $tasks$ by max variant size
   # Allocate variants with residency constraints
3: for $task$ in $T_{max}$ do
4:     $V_{min}$ = sort $variants$ of $task$ by min variant size
5:     for $variant$ in $V_{min}$ do
6:         if $variant$ has residency constraints AND $task$ has no $variant$ assigned then
7:             Allocate $variant$ to $processor$ from $R^{task}_{variant}$
   # Allocate variants with coresidency constraints
8: for $task$ in $T_{max}$ do
9:     if $task$ has coresidency constraints AND $task$ has no $variant$ assigned then
10:        Allocate smallest $variant$ to $processor$ from $P_{max}$
   # Allocate remaining variants
11: for $task$ in $T_{max}$ do
12:     if $task$ has no $variant$ assigned then
13:         Allocate smallest $variant$ to $processor$ from $P_{max}$
   # Upgrade variants where possible
14: while there are $task$ with $variants$ to explore do
15:     for $task$ in $T_{max}$ do
16:         if sufficient $capacity$ in assigned $processor$ then
17:             Allocate next larger $variant$ of $task$
18: if all $tasks$ assigned then
19:     return $allocation$
```

Algorithm 2 Local Search Metaheuristic

```
1: $current$ ← random assignment
2: while time < $timeout$ do
3:     for $n$ in neighbours($current$) do
4:         if $n$ is superior to $current$ then
5:             $current$ ← $n$
6:     if no improvement then
7:         $current$ ← random assignment
```

3.3 Local search metaheuristic

The third algorithm we propose is a simple local search metaheuristic employing random restarts. The process is described by Algorithm 2. Initially, a random assignment is generated by allocating a random variant for each task to a random processor (line 1), and all choices are made uniformly random. There is no guarantee a randomly generated allocation will satisfy the constraints of the model, and indeed the search algorithm is not guaranteed to find a feasible solution. Neighbouring solutions are generated (line 3) and accepted if the resulting allocation is superior to the incumbent one (lines 4–5). As there is no way to determine if the global optimum has been found, the algorithm continues to search the space of assignments until a given `timeout` is reached. The search may find a local optimum, in which case a random restart is used to explore other parts of the search space (lines 6–7).

The neighbourhood of a solution in the space of allocations is defined as all those solutions that can be generated by substituting another variant of the same task for one already allocated, or alternatively by moving a single variant to a different processor. In order to determine if one solution is preferable to another, a priority ordering amongst the constraints and objectives is established, in order of importance:

1. No processors should be overloaded.
2. The network should not be overloaded.
3. Residency constraints must be satisfied.
4. Coresidency constraints must be satisfied.
5. Average QoS per task should be maximised.
6. Average free capacity per CPU should be maximised.

A solution is feasible if the first four constraints are satisfied, after which the search will try to optimise QoS and then reduce processor utilisation to free up capacity. This priority ordering method is preferred over the alternative of a *weighted sum objective*, an approach found elsewhere in the literature (Marler and Arora 2009). Weighted sum approaches require the user to precisely quantify the relative importance of objectives and constraints, which is a somewhat inelegant approach to this problem, as it can be unrealistic in many scenarios (e.g. when considering factors such as execution time, energy consumption, and functional performance). Furthermore, it is known that some members of the pareto front will not be found when using such an approach (Coello et al. 2006; Das and Dennis 1997). As we prioritise functional performance over non-functional concerns, a two-stage approach is more appropriate. For the same reason, we prefer local search over *simulated annealing* (Tindell et al. 1992), an algorithm we also experimented with, which relies on a numerical gradient in the constrained objective space as a measure of absolute quality (i.e. it requires to provide weightings in the same manner as a weighted sum approach, which suffers from the problems given above).

3.4 Computational complexity

In the worst case, hill-climbing has time complexity $O(\infty)$, i.e. it may never find the global optimum. Our implementation of hill-climbing (i.e. local search metaheuristic) uses the *random restart* strategy, that is, it commences a new search once it has found a local optima, but this still does not guarantee it will find the global optimum. Similarly, greedy-search cannot provide such a guarantee. Finally, the worst-case complexity of the constraint programming approach is in principle exponential, but our results show that this does not occur in practice on the datasets analysed.

4 Example case study

Our case study serves as a specific instantiation of the general formulation presented, with which we can test our algorithmic solutions in a real system. We first present a *baseline*

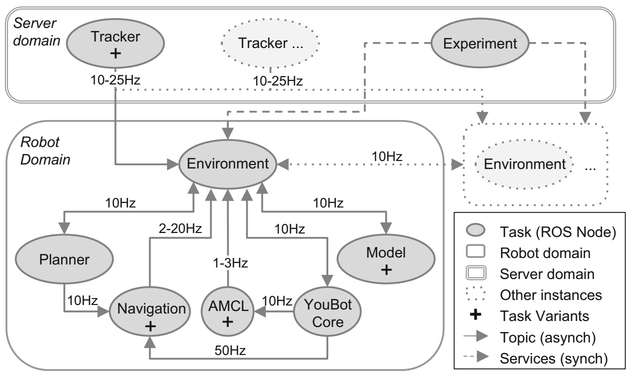


Fig. 2 Case study software architecture, composed of one *Tracker* instance per camera, one instance of each task in the *Robot domain* per robot, and one *Experiment* instance for the complete system

instance of the system, consisting of a single robot, person, server and camera. This simplified configuration illustrates the system components and the constraints imposed on them. Each robot or human agent is pursuing a spatial goal. The application's overarching QoS metric is a combination of essential requirements (e.g. avoid collisions between agents, minimise travel time to reach target goals), as well as more sophisticated preferences (e.g. minimise close-encounters and hindrance between navigating agents, minimise the time taken to infer the true agent goal). Therefore, task variants must be selected and allocated across available processors with the objective of optimising global QoS based on the selected variants' individual QoS values.

4.1 Software architecture

Figure 2 shows a high-level diagram representing the software architecture of the case study. It is composed of multiple tasks and their message connections. In the figure, connections are labelled with message frequencies, which can be obtained from the maximum bandwidth requirement described in Sect. 2.1. The QoS values for the variants of a given task represent the proportional benefit of running that task variant; a variant that has a higher QoS, however, would typically incur a higher CPU usage. We rely on an expert system user to estimate QoS values for task variants.

We now describe for each task in our case study, the corresponding variants (see Table 1 for details):

- *Tracker* A component of a distributed person tracking algorithm that fuses multiple-camera beliefs using a particle filter. The variants for this task are based on the input image resolution and the output frame rate given a fixed number of cameras. The higher the output frame rate the more accurate the tracking.
- *Experiment* A small synchronous task that coordinates all robots taking part in the experiment.

Table 1 Task variants characterisation

Task	Variants	Parameters	CPU	Freq (Hz)	BW (KB/s)	Res	CoRes	QoS
Experiment	1	–	1	10	1	Server	–	1
Tracker	4	Output freq. (25 20 15 10)	200 160 120 80	25 20 15 10	2.5 2 1.5 1	Server	–	100 90 70 40
Environment	1	–	1	10	0.5	–	–	1
Model	3	Num. goals (10000 3500 4)	59 39 17	10 10 10	5 5 5	–	–	100 60 20
Planner	1	–	1	10	0.5	–	Navig.	1
AMCL	3	Particles (3000 500 200)	66 41 19	2.5 2.5 2.5	1 1 1	–	–	100 75 50
Navigation	3	Controller freq. (20 10 2)	50 39 25	20 10 2	1 0.5 0.1	–	Planner	100 67 33
Youbot_Core	1	–	16	10	0.5	Robot	–	1

– *Environment* A local processing task required by each robot. This task combines information generated by the local robot, other robots, and elsewhere in the system (i.e. *Tracker*, *Experiment*).
– *Model* An intention-aware model for predicting the future motion of interactively navigating agents, both robots and humans. The variants for this task are based on the number of hypothetical goals considered given a fixed number of agents. A higher number of modelled agent goals will lead to more accurate goal estimates.
– *Adaptive Monte Carlo Localisation (AMCL)* Localisation relying on laser data and a map of the environment (Pfaff et al. 2006). The variants of this task vary with the number of particles used during navigation, since a larger number increases localisation robustness and accuracy in environments populated with other moving obstacles. We assume the robot moves on average at the preferred speed of 0.3 m/s (min 0.1 m/s, max 0.6 m/s).
– *Planner* Generates an interactive costmap, which predicts the future motion of all agents with relation to other agents' motion given their inferred target goals. Since the costmap is used by the *Navigation* task for calculating the trajectory to be executed, the two tasks have a coresidence constraint to guarantee a proper behaviour.
– *Navigation* This task avoids detected obstacles and attempts to plan a path given the interactive costmap of the agents in the environment, ultimately producing the output velocity the robot platform must take. The variants of *Navigation* depend on the controller frequency, that is, the number of times per second the task produces a velocity command. The higher the frequency, the more reactive and smooth the robot navigation becomes.
– *YouBot_Core* A set of ROS packages and nodes that enable the robot to function, for example *etherCAT* motor connectivity, internal kinematic transformations, and a laser scanner sensor. This task must always run in the corresponding robot (i.e. it has a residence constraint).

Finally, we assume that a robot is capable of executing a full set of its tasks, at the very least by selecting their least-demanding variants. Those tasks are represented within the robot domain in Fig. 2. This is critical to ensure a continued service in periods of network outage in future dynamic scenarios, albeit at lower levels of QoS.

4.2 Hardware architecture

The hardware integrating the baseline system is composed of a single network camera and two processors, that is, a robot with onboard processor and a remote server. Robot and server communicate through a wireless network, and camera and server through a wired network. In practice the network bandwidth is currently not a limiting factor, as both networks are dedicated and private in our lab. The same applies to the latency/quality of the wireless signals.

5 Evaluation

In this section, we first describe the results of an empirical characterisation of the baseline system, which is mandatory to evaluate both the solution methods and the case study itself. We then extend this characterisation to define a set of system instances of increasing size and complexity. Having established these benchmark problems, we employ them to evaluate the utility of our solution methods, in two stages.

In the first stage, we compare the quality of solutions returned by the three proposed methods to answer the following research questions:

– *RQ1A* Is it possible to find globally optimal variant selections and allocations using constraint programming?
– *RQ1B* How well can a straightforward greedy heuristic and the local search metaheuristic perform on this problem, relative to the constraint programming method?
– *RQ1C* How does the solution time scale with the size and complexity of the example system instances?
– *RQ1D* How well do the results produced by the three solution methods translate to deployment on the physical system outlined in Sect. 4?
– *RQ1E* How effective are the allocations proposed by our solution methods compared to random allocations?

In the second stage, we compare the constraint programming solver configured in *anytime* mode against the local search metaheuristic, to explore their performance over time. Our research questions are as follows:

– *RQ2A* How do local search metaheuristic and "anytime" constraint programming compare in terms of their solutions quality after a given period of run-time?
– *RQ2B* Could these two "anytime" methods be used in future dynamic scenarios?

5.1 System characterisation

We performed an offline characterisation of the baseline system shown in Fig. 2 using common monitoring utilities from ROS (e.g. *rqt*, which provides average values) and Linux (e.g. *htop*, visually inspecting it during execution). The objective was to obtain for each unique task variant in the system the following values: (i) the average percentage of CPU utilisation required; (ii) the average frequency at which messages published are sent to other tasks; and (iii) the average network bandwidth required.

Table 1 summarises the values obtained. Column two represents the number of variants for each task, and column three the value of the parameters that create the task variants (see Sect. 4.1). The next three columns include the average values of CPU utilisation, frequency and bandwidth for each task variant—note that the maximum values for frequency are shown in Fig. 2. The CPU values for the *Tracker* task assume only one person in the environment. Columns seven and eight show the residence and coresidence constraints for each variant and task respectively. Finally, the last column represents the normalised QoS associated with each task variant, where 100 is the maximum value. Note that we have assigned QoS value "1" to single variant tasks because they have much less impact in the system behaviour, which is reflected in low CPU utilisation values in Table 1. The focus of this work is task variant allocation, for which we require QoS values as inputs. Although QoS values were manually generated based on real system measurements, they may be automatically generated, but we leave this for future work. It is worth noting that the user is required to provide QoS values only "once" for each task variant. Therefore, when the system is scaled up by replicating tasks on more robots or cameras, no further manual characterisation work is required from the user.

Finally, we specify the hardware characteristics of the baseline system. The robot's on-board processor is an Intel Atom, 2 cores @ 1.6 GHz, 2 GB RAM. The server's processor is an Intel i5-3340, Quad Core @ 3.30 GHz (Turbo), 16 GB RAM. Note that all CPU measurements are normalised to the robot CPU capacity (assumed as 100). From this, we can understand why the *Tracker* instances (which have a high CPU requirement) can only run in the server, translating into a residence constraint. The networks employed are a wireless IEEE 802.11ac network at 300 Mbps, and a wired 1 Gbps Ethernet network.

5.2 System instances

In order to obtain more complex instances of the baseline system shown in Fig. 2, we only need to add processors (i.e. robots, servers) and/or cameras, allowing the system to cope with a more complex environment and complete more difficult challenges. As these parameters are varied, the total number of tasks and variants changes accordingly, but the number of variants for each task is fixed.

Table 2 summarises the set of instances comprising our benchmarks, which includes the number of tasks and variants generated for each case—note that only one server with a capacity of 400 is used for all cases. In order to provide an estimate of the search space size, an approximate upper bound N_k for the number of possible variant assignments for a given problem instance k is calculated as follows:

Table 2 System instances considered

Inst.	Proc.	Robots	Cam	Tasks	Var.	S. space N_k
1	2	1	1	8	17	1728
2	2	1	2	9	21	6912
3	2	1	3	10	25	27,648
4	3	2	1	14	29	1.91×10^5
5	3	2	2	15	33	7.65×10^5
6	3	2	3	16	37	3.06×10^8
7	4	3	1	20	41	1.32×10^{12}
8	4	3	2	21	45	5.28×10^{12}
9	4	3	3	22	49	2.11×10^{13}
10	4	3	4	23	53	8.45×10^{13}
11	5	4	1	26	53	4.45×10^{14}
12	5	4	2	27	57	1.78×10^{15}
13	5	4	3	28	61	7.12×10^{15}
14	5	4	4	29	65	2.85×10^{16}

$$N_k = \prod_{\tau_i \in T} |P(\tau_i)| \cdot |\tau_i| \tag{8}$$

where $|P(\tau_i)|$ is the number of possible processors on which a task τ_i can be allocated without violating residency and coresidency constraints, and $|\tau_i|$ is the size of the set of variants of the task. The size of the search space N_k for the instances considered is shown in the last column of Table 2.

5.3 Simulation results

We now analyse and compare the QoS and CPU utilisation values of solutions provided by the three proposed methods (since these are simulation results,[2] we call them expected values). Remember that the allocation of more powerful variants translates into higher global QoS values, and strongly correlates with improved overall system behaviour. For example, switching from the least to most powerful variant of the $Tracker$ task (QoS values 40 and 100 in Table 1) actually provides more accurate and faster tracking of people in the environment. This in turn provides the $Planner$ and $Model$ tasks with better data, improving the robots ability to navigate (e.g. avoiding collisions).

We execute Python programs implementing the three proposed methods for the instances described in Table 2. All simulation experiments were performed on a 2.8 GHz Intel Core i7 with 4 GB RAM (Table 3 shows the total execution times). Answering *RQ1A*, we found that constraint programming finds the globally optimal solution for all instances analysed. In other words, for each instance this method pro-

[2] Note that we simulate the QoS and CPU loads given the characterised values in Table 1. No physics/robot simulation was used.

Table 3 Total execution time of greedy heuristic (GH), local search (LS), and constraint programming (CP) for the system instances considered

Instance	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GH	3 ms	5 ms	6 ms	10 ms	13 ms	14 ms	21 ms	23 ms	22 ms	25 ms	30 ms	29 ms	33 ms	37 ms
CP/LS	340 ms	340 ms	390 ms	1.36 s	2.34 s	3.54 s	14.20 s	9.6 m	33 m	17 m	10.1 h	6.19 d	2 w	6.81 d

ms milliseconds, *s* seconds, *m* minutes, *h* hours, *d* days, *w* weeks

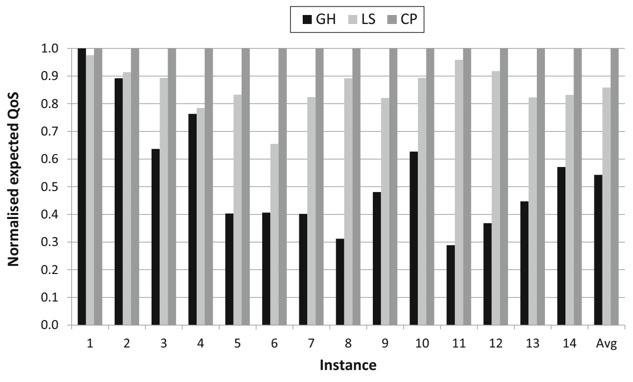


Fig. 3 Expected QoS for greedy heuristic (GH), local search (LS), and constraint programming (CP). Values are normalised to the optimal solution (= 1). Server capacity = 400

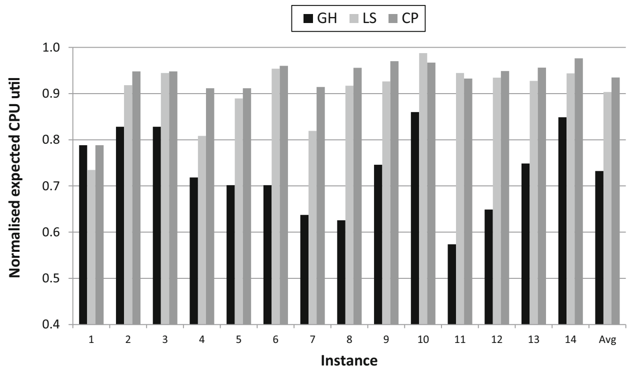


Fig. 4 Expected CPU utilisation for greedy heuristic (GH), local search (LS), and constraint programming (CP). Values are normalised to the corresponding total CPU capacity (= 1). Server capacity = 400

vides the allocation of task variants to processors with the best possible average QoS and minimum CPU utilisation. Since constraint programming provides the best possible QoS, we normalise the QoS provided by the greedy and local search methods to the optimum. Figure 3 shows results comparing the QoS of the three methods—note that values for local search are actually the average of three independent runs with a timeout set to be equal to the time required by the constraint programming method. Therefore, answering *RQ1B*, we observe that local search and greedy heuristic achieve an average of 14 and 46% lesser QoS than constraint programming respectively.

Figure 4 shows the results for CPU utilisation, where the values indicated refer to the total utilisation of the sum of all CPU capacities. On average, constraint programming utilises 3 and 20% more CPU capacity than local search and greedy respectively, but as shown in Fig. 3, the differences in QoS are much greater (14% and 46%). There are two special cases in Fig. 4 where local search utilises more CPU capacity than constraint programming while providing lesser QoS. For *Instance* 10, the reason is that local search finds an infeasible solution. However for *Instance* 11 the solution found is feasible, which further demonstrates that constraint programming provides better solutions—recall that it uses a two-pass method.

Since we maintain the server capacity (= 400) across all instances analysed, the problem becomes more constrained as the total number of task variants increases. As an example, constraint programming and the greedy heuristic solve *Instance* 1 allocating the most powerful variant for all tasks

Table 4 *Instance* 1 (1 robot, 1 camera): task variant selection (V is the variant, smaller numbers indicate more powerful) and allocation (P is the processor, biggest number is the server) for each solution method

Tasks (8)	V_{GH}	P_{GH}	V_{LS}	P_{LS}	V_{CP}	P_{CP}
Experiment	1	2	1	2	1	2
Tracker	1	2	3	2	1	2
Environment	1	2	1	2	1	1
Model	1	2	1	1	1	2
Planner	1	2	1	2	1	1
AMCL	1	2	1	2	1	2
Navigation	1	2	1	2	1	1
Youbot_Core	1	1	1	1	1	1

($V_{CP} = 1$ and $V_{GH} = 1$ in Table 4). Note how constraint programming balances much better the allocation of task across the available processors (P_{CP} and P_{GH} in Table 4). However for *Instance* 10 (Table 5), some tasks need to use less powerful variants in order to satisfy the CPU capacity constraint (e.g. the four *Tracker* tasks use the least powerful variant for constraint programming, $V_{CP} = 4$). In Table 5 we also see how local search allocates *Tracker_1* to *Processor* 3 ($P_{LS} = 3$), thus providing the infeasible solution commented previously. Since the *Tracker* task has a residence constraint, it can only be allocated to the server, which is *Processor* 4 for this instance.

Finally, and answering *RQ1C*, we see that the solution times for constraint programming are reasonable up to *Instance* 10 (Table 3), which is actually a big system in

Table 5 *Instance* 10 (3 robots, 4 cameras): task variant selection (V is the variant, smaller numbers indicate more powerful) and allocation (P is the processor, biggest number is the server) for each solution method

Tasks (23)	V_{GH}	P_{GH}	V_{LS}	P_{LS}	V_{CP}	P_{CP}
Experiment	1	4	1	4	1	4
Tracker_1	4	4	4	**3**	4	4
Tracker_2	4	4	3	4	4	4
Tracker_3	4	4	4	4	4	4
Tracker_4	4	4	4	4	4	4
Environment_1	1	2	1	3	1	1
Model_1	2	3	3	3	2	3
Planner_1	1	4	1	1	1	2
AMCL_1	3	2	2	1	3	3
Navigation_1	3	4	2	4	1	2
Youbot_Core_1	1	1	1	1	1	1
Environment_2	1	4	1	2	1	1
Model_2	3	2	1	4	2	4
Planner_2	1	4	1	2	1	1
AMCL_2	3	2	1	2	3	2
Navigation _2	3	4	3	4	2	1
Youbot_Core_2	1	2	1	2	1	2
Environment_3	1	2	1	3	1	1
Model_3	2	3	3	1	2	4
Planner_3	1	4	1	3	1	1
AMCL_3	3	2	3	4	3	3
Navigation_3	3	4	3	1	2	1
Youbot_Core_3	1	3	1	2	1	3

terms of the search space (Table 2). At this point, the times start to become intractable (we analyse the anytime behaviour in Sect. 5.5). Thus, we don't report results for larger system instances. We leave further refinements to our MiniZinc model as future work, which could potentially allow scaling to larger instances. On the other hand, the solution times for the greedy heuristic are small (milliseconds) and scale well, although it provides inferior QoS values to constraint programming, as previously discussed.

5.4 Analysis of case study behaviour

Having obtained the simulation results, our next step is to validate that the expected QoS values obtained via simulation match the behaviour of the real system. To do this, we performed experiments for instances 1–6 from Table 2 in our case study environment. For each instance, we configured the allocation of task variants to processors computed by the solution methods—note that only a single human agent is present in the environment for all experiments. Then, the measured QoS value for each instance and method is obtained

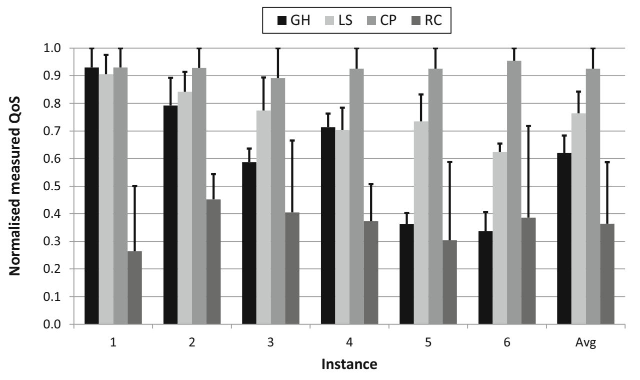


Fig. 5 Validation of simulation results on the physical system: measured QoS for greedy heuristic (GH), local search (LS), constraint programming (CP), and random allocations (RA). Error bars represent the deviation from the expected QoS values

by applying the following formula:

$$QoS_{measured} = \sum_{\tau \in T} QoS_\tau \times \frac{F^o_\tau}{F^e_\tau} \tag{9}$$

where QoS_τ is the expected QoS value for task τ as predicted by our solution methods, F^o_τ is the observed frequency of messages produced by task τ on the real system and F^e_τ is the expected frequency associated with task τ (Table 1). These two frequencies can differ due to overloaded processors (for infeasible solutions) and/or approximation errors in the system characterisation. Therefore, this frequency ratio determines the effectiveness of a task variant in the real system.

Figure 5 shows the results. The black error bar for each column denotes the difference between measured QoS (top of column) obtained with Eq. 9, and expected QoS (error bar upper end) obtained by simulation. Answering *RQ1D*, the measured QoS values for local search, constraint programming and the greedy heuristic only deviate by 8, 7 and 5% on average respectively from the expected values. This result validates the accuracy of our methodology.

Finally, we also examined the system behaviour considering random allocations of task variants to processors—note that this could be the best choice for users with little knowledge of the system or for large system instances. Figure 5 also includes these results (RA), where each bar actually corresponds to the average QoS of three randomly generated allocations. Answering *RQ1E*, we see how the measured QoS values for random allocations deviate much more from the expected ones, by an average of 22%, than those for the proposed solution methods. The reason is that our solution methods produced feasible allocations for the six instances analysed (i.e. satisfying system constraints), thus differences are only due to approximation errors in the system characterisation. However, some of the random allocations produced infeasible solutions, which translated into overloaded processors and therefore larger differences with the expected values.

In summary, constraint programming improves the overall QoS of the real system by 16, 31, and 56% on average over local search metaheuristic, greedy heuristic and random allocations respectively.

5.5 Anytime approaches

In this section, we consider the task allocation problem from a different point of view. If we are to apply our approach to larger systems, or select and allocate task variants at system boot, or even at run-time, then the time taken to find a good allocation takes on greater precedence. Both the *Gecode* constraint programming solver and local search solution methods can be used as *anytime algorithms*, where the best allocation currently known can be returned at any point during their execution. The two algorithms approach the problem differently, because constraint programming requires a two-pass procedure where each objective is optimised in turn, whereas the local search metaheuristic attempts to optimise both objectives simultaneously. Therefore, the relative performance of the two algorithms is of interest.

For our *anytime* analysis, we selected the instances that resulted in significant runtimes for constraint programming (Table 3), that is, from $Instance$ 7 to $Instance$ 14. Then, we executed the *Gecode* solver and the local search method for these instances with increasing timeout values, to evaluate how the solutions they found improved over time. Figure 6b–i show the results. The graphs show two objectives: firstly, the *Quality of Service* objective as defined by Eq. 1, and secondly the *Utilisation* objective as defined by Eq. 2. Each intermediary result is from an independent run of the algorithms, avoiding the problem of autocorrelation. All results in Fig. 6 are normalised to the optimal solution ("1"), which represents: (i) for QoS, the best possible value; (ii) for CPU utilisation, unutilised processors (free capacity of 100%). Note that for $Instances$ 11 to 14 we only show the execution time for the first 4000 s, but in all cases the QoS obtained is greater than the 80% of the optimal value.

As can be observed for all the instances in Fig. 6, there is a certain amount of variance in the results produced by local search, based on the seed provided. For example, the fifth value for local search QoS in Fig. 6b is lower than the preceding and following values. To measure the variance for this instance, we repeated the experiment ten times, and present the results in Fig. 6a. This underlines the fact that the performance of local search is quite variable, although it generally makes steady progress over time.

The graphs in Fig. 6 illustrate a clear trend that answers *RQ2A*: in most cases, constraint programming produces superior results in the same amount of time, and is our preferred anytime solution method. The only exception in our results is $Instance$ 12, where local search provides better QoS values for the last six points of the graph (Fig. 6g)—note that heuristic-based methods can randomly provide a good solution. For the first point ($time = 10$ s) local search provides an infeasible solution. The other instances containing points where local search is better than constraint programming (i.e. Fig. 6d, e, h, i) also correspond to infeasible solutions.

Answering *RQ2B*, constraint programming also produces high quality results within a short timeframe, which may enable dynamic optimisation in the future and also increases our confidence in its ability to scale to larger systems. Figure 7 shows the first feasible solution provided by constraint programming normalised to the optimum. This first result is provided after 4 s for $Instances$ 7–10, and after 10 s for $Instances$ 11–14. The figure also shows the value for the greedy heuristic, just to have a clear idea of how good constraint programming is for these short times, a 32% better on average. Note that local search is not shown because after 10 s it provides infeasible solutions for almost all the instances.

Finally, note that since our local search algorithm is implemented in Python, it may be argued that constraint programming has an unfair advantage in that the MiniZinc solvers are written in C; however, the highly optimised nature of constraint solvers is actually a strong argument in favour of adopting them, particularly as they improve through continuous development over time.

6 Related work

Much work has been performed in the area of task allocation in distributed robotics, where different types of optimisation problems have been addressed. A comprehensive taxonomy can be found in Korsah et al. (2013), where problems are categorised based on: (i) the degree of interdependence of agent-task utilities; and (ii) the system configuration, which in turn is based on an earlier taxonomy Gerkey and Matarić (2004) that considers the type of: agents, tasks and allocation. According to these taxonomies, the task variant allocation problem presented in this paper falls in the category of Cross-schedule Dependencies (XD), that is, the effective utility of each individual task-agent allocation depends on both the other tasks an agent is performing, and the tasks other agents are performing. Several types of system configurations are supported within this category—e.g. MT–SR–IA considers multi-task robots (MT), single-robot tasks (SR), and instantaneous task assignment (IA). Furthermore, problems in this category can be formulated with different types of mathematical models. In our case, we use a special form of knapsack formulation (Sect. 2).

Section 6.1 presents a systematic literature review of the area. Our key goals are to:

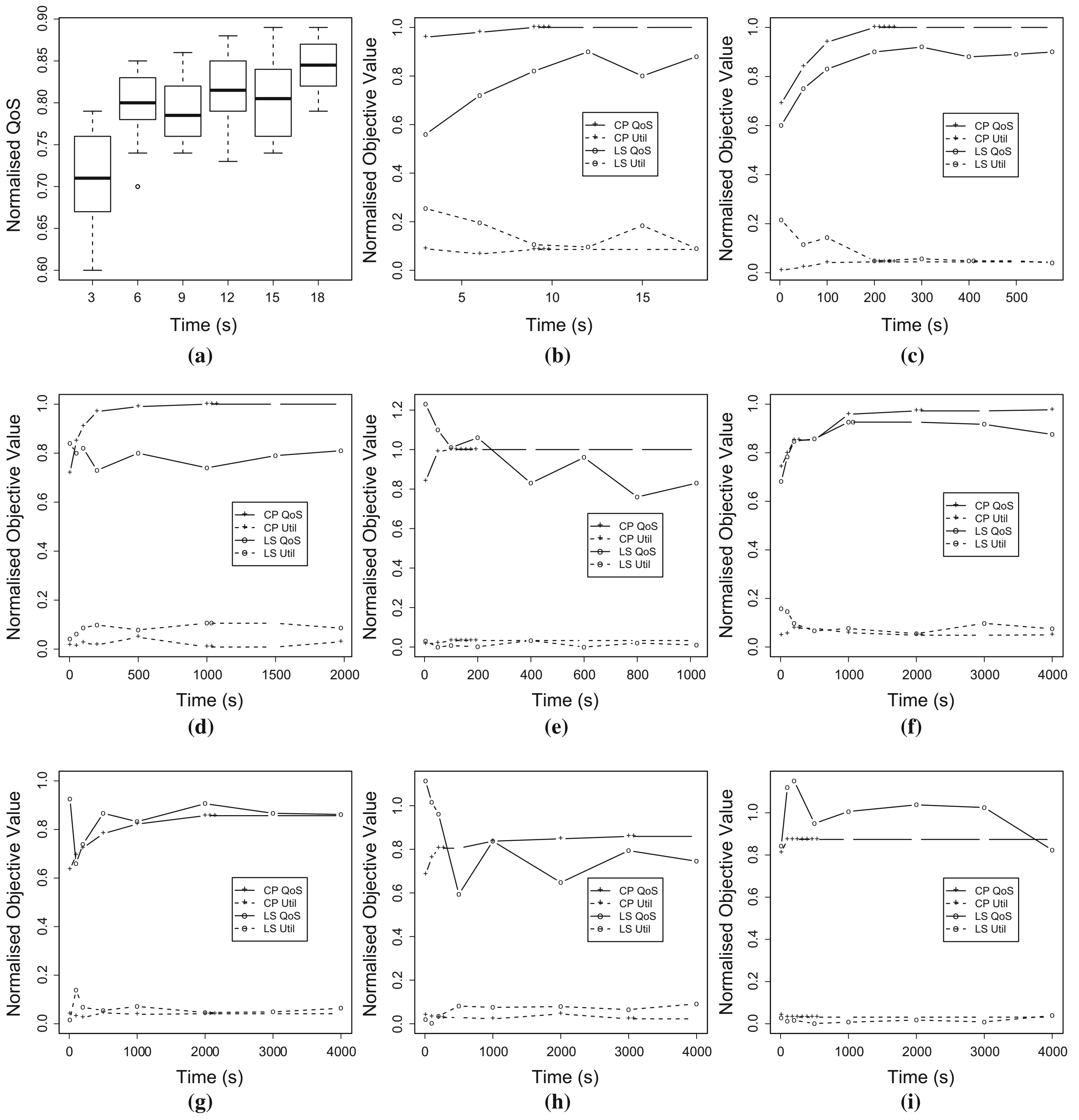


Fig. 6 Anytime results for constraint programming (CP) and local search (LS): **a** Distribution on *Instance* 7 with 10 Repetitions; **b** *Instance* 7; **c** *Instance* 8; **d** *Instance* 9; **e** *Instance* 10; **f** *Instance* 11; **g** *Instance* 12; **h** *Instance* 13; **i** *Instance* 14

1. Quantify the size of typical search spaces for robot task allocation case studies.
2. Characterise the general solution methods used for task allocation in this domain.

This survey shows that our problem search spaces are larger than previous work, and our constraint programming optimisation technique is novel.

Section 6.2 proceeds to study key related work in distributed robotics falling in the same category as our work. We highlight how our work differs from past research.[3] In systems like our case study, where task variants are instantiated by parameter configurations, it might be interesting to

[3] Note that this paper is an extension of our previous work (Cano et al. 2016b).

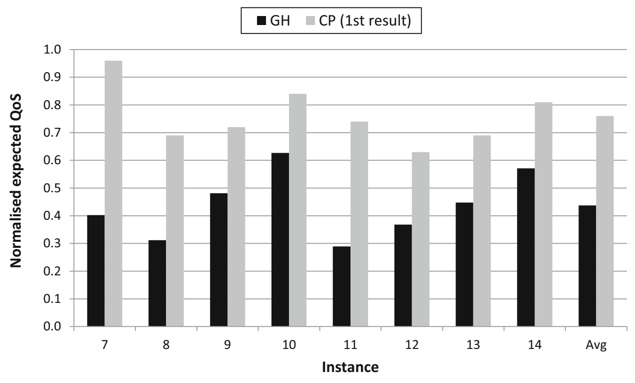


Fig. 7 Expected QoS for for greedy heuristic (GH), and the first result provided by constraint programming (CP 1st result). Values are normalised to the optimal solution (=1). Server capacity = 400

consider a greater number of variants for each task (even a continuum set), but then the allocation problem would require different solution methods, as shown in Cano et al. (2016a).

6.1 Systematic review

This section presents a systematic review of the literature, following standard practice in the discipline (Kitchenham 2004). The research is performed to establish the search space boundaries in optimisation of task allocation for robotic applications. The survey was conducted so that we can quantitatively assess the maximum experiment size (either actual or simulated) of each published work included. This approach provides a comparative base between the scalability of existing work and our method in practice. In that respect we attempt a characterisation of limitations of task allocation approaches in the domain in comparison with the limitations of our approach.

Initially, three types of robotic systems were identified. The first type involved a larger number of robots performing less demanding tasks such as swarm systems. The second involved a moderate number of robots (10–30) (Sugiyama et al. 2016; Cheng et al. 2016). The third included a low number of robots performing very complex tasks (Khalfi et al. 2016; Wang et al. 2016; Hongli et al. 2016).

The strategy used for the systematic review involved searching for journal and conference publications through three well established databases and with a predefined combination of keywords. The databases are *IEEE Xplore* (IEEE 2016), *ScienceDirect* (Elsevier 2016) and *ACM Digital Library* (The Association for Computing Machinery 2016). Moreover, proceedings of well known conferences were reviewed and in particular *IROS, AAMAS, ICRA* and *RSS*. The search pattern used was *robot AND ("task allocation" OR "resource allocation") AND (optimisation OR optimization)*. Moreover, to limit the search only the first 10 pages of results were examined for each of the databases. Further we restricted publications based on inclusion, exclusion and quality criteria below.

The inclusion criteria for published articles to be reviewed were:

1. Complex tasks.
2. Independently operating robots.
3. Task allocation either automated or manual.

The exclusion criteria were:

1. Simplistic tasks.
2. Small robots e.g. ant, swarm robots operating as a crowd.
3. Literature review and survey papers not describing evaluation approaches.
4. Papers where the experimental setup was not clearly presented.

Each paper was assessed for inclusion by two of the authors. Disagreements were arbitrated by a third author, at which point the decision was final. Additionally, to restrict the search scope we established quality criteria. As the extracted information relates to experimental setup and not results of any of the reviewed articles, no publication bias is expected and thus no measures were designed in this protocol to avoid it. On the other hand, only conferences included in the highest rank of three ranking systems were included (AMiner 2017; Wirtschaftsinformatik 2017; CORE 2017). By looking at three ranking systems we attempt to remove bias and error from the ranking method. The ranking systems were accessed online on 19 January 2017. For journal publications a 2 year JCR impact factor higher than 2 was selected as a qualitative criterion for selection.

The information extracted from each paper is:

1. Number of robots used in the experiment.
2. Total number of tasks to be allocated.
3. Number of tasks per robot.
4. Allocation method.

Allocation methods were categorised as static when they were performed ahead of time or dynamic when tasks were allocated in the duration of the experiment. Moreover, the methods were categorised as centralised or distributed. Each publication was then characterised based on the extended taxonomy found in Korsah et al. (2013). From this information the search space N_{max} was calculated for each publication using Eq. 8, based on the largest experiment or maximum possible search space. This formula was simplified for auction and negotiation methods to:

$$N_{max} = n \times \tau^2 \tag{10}$$

Table 6 Systematic review for robot task allocation. *Plus publications reviewed in Cano et al.* (2016b)

Reference	Category (cf. Korsah et al. (2013))	Search Space (N_{max})	Allocation	Method
Kim et al. (2012)	ND–MR–ST–IA	1.6×10^5	Dynamic/simulated	Auction, distributed
Sikora et al. (2017)	ND–MR–ST–TA	1.7×10^5	Static–dynamic/actual	Sub-optimal, time limited, centralised
Prescott et al. (2006)	ID–SR–MT–IA	1.6×10^2	Dynamic/actual	Neural networks, distributed
Szomiki (2015)	ID–SR–MT–IA	2.6×10^2	Dynamic/simulated	Agent, distributed
Girard et al. (2008)	ID–SR–MT–IA	4.4×10^4	Static/simulated	Neural networks, distributed
Franceschelli et al. (2013)	ID–SR–MT–IA	5.0×10^6	Static+dynamic/simulated	Gossip algorithm, centralised + distributed
Nam and Shell (2015)	XD–SR–ST–IA	2.7×10^1	Static/simulated	Ranking algorithm/hungarian method, centralised
Liu and Shell (2012)	XD–SR–ST–IA	2.2×10^2	Dynamic/actual	Hungarian algorithm, centralised
Guo and Zhang (2010)	XD–SR–ST–IA	2.1×10^4	Dynamic/simulated	AI/auction, distributed
Hu and Xu (2013)	XD–SR–ST–IA	4.7×10^4	Dynamic/simulated	Cooperative control, distributed
Nam and Shell (2015)	XD–SR–ST–IA	1.0×10^6	Static/actual	Ranking algorithm/hungarian method, centralised
Liu and Shell (2011)	XD–SR–ST–IA	3.4×10^6	Dynamic/simulated	Multi-level partitioning, distributed
Liu and Shell (2012)	XD–SR–ST–IA	2.7×10^7	Dynamic/simulated	Hungarian algorithm, centralised
Liu and Shell (2013)	*XD–SR–ST–IA*	*1.0×10^9*	*Dynamic/simulated*	*Auction, distributed*
Liu and Shell (2012)	*XD–SR–ST–IA*	*1.0×10^{12}*	*Dynamic/simulated*	*Swap, distributed*
Suemitsu et al. (2016)	XD–SR–ST–TA	1.0×10^2	Static/actual	Genetic algorithm, centralised
Chu and ElMaraghy (1993)	XD–SR–ST–TA	1.6×10^4	Static/actual	Heuristic, centralised
Caraballo et al. (2017)	XD–SR–ST–TA	8.0×10^4	Dynamic/simulated	Negotiation, distributed
Dias and Stentz (2002)	XD–MR–ST–IA	8.0×10^2	Static/simulated	Market-trading, distributed
Giordani et al. (2013)	XD–MR–ST–IA	7.5×10^5	Dynamic/simulated	Auction/hungarian method, distributed
Chen and Sun (2011)	XD–MR–ST–IA	6.3×10^3	Static+dynamic/actual	Coalition-based, distributed
Zhang and Parker (2013)	*XD–MR–ST–IA*	*7.4×10^9*	*Dynamic/simulated*	*Heuristic, distributed*
Parker and Gini (2014)	XD–MR–ST–TA	1.0×10^4	Dynamic/simulated	Heuristic, distributed
Agarwal et al. (2015)	XD–MR–ST–TA	6.3×10^4	Dynamic/simulated	Coalition based, centralised
Miyata et al. (2002)	*XD–SR–MT–IA*	*3.2×10^2*	*Static/actual*	*Priority-based, centralised*
Lagoudakis et al. (2004)	XD–SR–MT–IA	1.2×10^3	Dynamic/simulated	Auction, distributed
Lozenguez et al. (2013)	XD–SR–MT–IA	1.6×10^4	Dynamic/simulated	Auctions/markov decision, distributed
Sujit et al. (2006)	XD–SR–MT–IA	1.8×10^4	Dynamic/simulated	Negotiation, distributed
Zhao et al. (2016)	XD–SR–MT–IA	6.6×10^4	Dynamic/simulated	Heuristic, distributed
Luo et al. (2015)	*XD–SR–MT–IA*	*1.4×10^6*	*Dynamic/simulated*	*Auction, distributed*
Nunes et al. (2012)	XD–SR–MT–IA	2.5×10^6	Static/simulated	Auction, distributed
Mosteo and Montano (2007)	XD–SR–MT–IA	5.2×10^6	Dynamic/simulated	Auction, distributed
Tolmidis and Petrou (2013)	XD–SR–MT–IA	7.2×10^7	Dynamic/simulated	Auction/genetic algorithm, centralised + distributed

Table 6 continued

Reference	Category (cf. Korsah et al. (2013))	Search Space (N_{max})	Allocation	Method
Okamoto et al. (2011)	XD–SR–MT–IA	8.0×10^8	Dynamic/simulated	Peer-to-peer token passing, distributed
Balakirsky et al. (2007)	*XD–SR–MT–TA*	*2.0×10^3*	*Dynamic/simulated*	*Auction, distributed*
Zorbas et al. (2016)	XD–SR–MT-TA	1.3×10^5	Static/simulated	Greedy heuristic, centralised

Italics represent to the original conference paper

where τ is the number of tasks that can be allocated, and n is the number of robots negotiating or bidding for tasks. Similarly, there is another simplified formula for Neural Network methods:

$$N_{max} = \prod_{i \in layers} nodes_i \tag{11}$$

where *layers* denote the layers of the Neural Network and $nodes_i$ the number of nodes in the ith layer. Finally, when an explicit formula was presented in a publication that formula took precedence as long as it respected the constraints in Eq. 8.

As a result of this systematic survey, Table 6 presents the papers reviewed and a summary of information extracted. The selection of articles was performed 16–31 January 2017 with the information extraction and review of articles performed by 15 February 2017. Entries are sorted primarily by taxonomic classification, then by search space size. The work presented in the earlier sections of this paper is classified as XD–SR–MT–IA. Thus, by comparison, the search space of our work is several orders of magnitude larger than the maximum search space presented in related work in this category (Lagoudakis et al. 2004; Lozenguez et al. 2013; Luo et al. 2015; Miyata et al. 2002; Sujit et al. 2006; Zhao et al. 2016; Nunes et al. 2012; Mosteo and Montano 2007; Tolmidis and Petrou 2013; Okamoto et al. 2011).

6.2 Comparative analysis

The purpose of the systematic survey was primarily to quantify the search space of prior robotic case studies. In this section, we provide a deeper comparison of our work with the most closely related previous research.

The first difference arises from the number of tasks and agents considered. Prior work based on the linear assignment problem (Pentico 2007) assumes a single task per agent (Nam and Shell 2015; Luo et al. 2015; Liu and Shell 2013, 2012). In our case, the number of tasks is equal to or greater than the number of agents (and the number of variants is greater still). A second point is related to the number of agents simultaneously completing tasks. In Chen and Sun (2011), Zhang and Parker (2013), Balakirsky et al. (2007) several agents are required, while in our work only one agent is completing each task. Another consideration is that our system is fully heterogeneous, i.e. all tasks and processors may be different. Some past work does assume heterogeneous tasks and multiple instances of every task (Miyata et al. 2002), but does not consider different variants of the same task, which is the principal addition to the problem here.

On the other hand, Aleti et al. (2013) provide a high-level general survey of software architecture optimisation techniques. In their taxonomy, our work is in the problem domain of design-time optimisation of embedded systems. We explore optimisation strategies that are both approximate and exact. We evaluate our work via both benchmark problems and a case study. In terms of the taxonomy in Aleti et al. (2013) our work is particularly wideranging.

Finally, Huang et al. (2012) consider the selection and placement of task variants for reconfigurable computing applications. They represent applications as directed acyclic graphs of tasks, where each task node can be synthesised using one of four task variants. The variants trade off hardware logic resource utilisation with execution time. Huang et al. use an approximate optimisation strategy based on genetic algorithms to synthesise the task graph on a single FPGA device.

To summarise, no existing work in the robotics field addresses all of the considerations that our proposal does, i.e. a constrained, distributed, heterogeneous system with more tasks than nodes and different variants for the tasks.

7 Conclusion

We have addressed a unique generalisation of the task allocation problem in distributed systems, with a specific application to robotics. We advocate the use of task variants, which provide trade-offs between QoS and resource usage by employing different parameter configurations, and/or algorithms, and/or taking advantage of heterogeneous hardware. We have presented a mathematical formulation of variant selection and allocation, and evaluated three solution methods on system instances obtained from a robotics case study. We conclude that constraint programming is the best solution method, being very effective in selecting and allocating variants such that QoS is maximised and resource usage minimised. In addition, we find that our solutions methods translate well to real systems, providing a useful tool for the system architect. We also analysed two solution methods in anytime mode, concluding that constraint programming might be used in dynamic scenarios. Finally, we performed a comprehensive literature survey on prior case studies and found that the maximum search space of our case study is much larger than those in previous work. We believe future work on constraint programming can further extend the boundaries of what has been handled in this work.

Acknowledgements This work was supported by the AnyScale Applications project under the EPSRC Grant EP/L000725/1, and partially by the EPSRC Grants EP/F500385/1 and EP/K503058/1, and the BBSRC Grant BB/F529254/1. We thank Ornela Dardha and Patrick Prosser for their valuable help in the problem formulation.

References

Agarwal, M., Agrawal, N., Sharma, S., Vig, L., & Kumar, N. (2015). Parallel multi-objective multi-robot coalition formation. *Expert Systems with Applications*, *42*(21), 7797–7811.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., & Meedeniya, I. (2013). Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, *39*(5), 658–683.

AMiner. (2017). Computer science. https://aminer.org/ranks/conf. Accessed January 19, 2017.

Balakirsky, S., Carpin, S., Kleiner, A., Lewis, M., Visser, A., Wang, J., et al. (2007). Towards heterogeneous robot teams for disaster mitigation: Results and performance metrics from RoboCup rescue: Field reports. *Journal of Field Robotics*, *24*(11–12), 943–967.

Bischoff, R., Huggenberger, U., & Prassler, E. (2011). KUKA youBot: A mobile manipulator for research and education. In *IEEE International conference on robotics and automation* (pp. 1–4).

Bordallo, A., Previtali, F., Nardelli, N., & Ramamoorthy, S. (2015). Counterfactual reasoning about intent for interactive navigation in dynamic environments. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 2943–2950). IEEE.

Cano, J., Bordallo, A., Nagarajan, V., Ramamoorthy, S., & Vijayakumar, S. (2016). Automatic configuration of ROS applications for near-optimal performance. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 2217–2223).

Cano, J., Molinos, E., Nagarajan, V., & Vijayakumar, S. (2015). Dynamic process migration in heterogeneous ROS-based environments. In *International conference on advanced robotics* (pp. 518–523).

Cano, J., White, D. R., Bordallo, A., McCreesh, C., Prosser, P., Singer, J., & Nagarajan, V. (2016). Task variant allocation in distributed robotics. In *Proceedings of robotics: Science and systems*.

Caraballo, L., Daz-Bez, J., Maza, I., & Ollero, A. (2017). The block-information-sharing strategy for task allocation: A case study for structure assembly with aerial robots. *European Journal of Operational Research*, *260*(2), 725–738.

Chen, J., & Sun, D. (2011). Resource constrained multirobot task allocation based on leaderfollower coalition methodology. *The International Journal of Robotics Research*, *30*(12), 1423–1434.

Cheng, Q., Yin, D., Yang, J., & Shen, L. (2016). An auction-based multiple constraints task allocation algorithm for multi-UAV system. In *International conference on cybernetics, robotics and control* (pp. 1–5).

Chu, H., & ElMaraghy, H. (1993). Integration of task planning and motion control in a multi-robot assembly workcell. *Robotics and Computer-Integrated Manufacturing*, *10*(3), 235–255.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2006). *Evolutionary algorithms for solving multi-objective problems (genetic and evolutionary computation)*. New York: Springer.

CORE. (2017). Computing research & education conference portal. http://portal.core.edu.au/conf-ranks/. Accessed January 19, 2017.

Das, I., & Dennis, J. E. (1997). A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural Optimization*, *14*(1), 63–69. https://doi.org/10.1007/BF01197559.

Dias, M. B., & Stentz, A. (2002). Opportunistic optimization for market-based multirobot control. In *IEEE/RSJ international conference on intelligent robots and systems* (Vol. 3, pp. 2714–2720).

Elsevier B. V. (2016). ScienceDirect digital library. http://www.sciencedirect.com/. Accessed January 23, 2016.

Franceschelli, M., Rosa, D., Seatzu, C., & Bullo, F. (2013). Gossip algorithms for heterogeneous multi-vehicle routing problems. *Nonlinear Analysis: Hybrid Systems*, *10*, 156–174. Special issue related to IFAC conference on analysis and design of hybrid systems (ADHS 12).

Gecode Team. (2006). Gecode: Generic constraint development environment. http://www.gecode.org. Accessed October 23, 2017.

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

Giordani, S., Lujak, M., & Martinelli, F. (2013). A distributed multi-agent production planning and scheduling framework for mobile robots. *Computers and Industrial Engineering*, *64*(1), 19–30.

Girard, B., Tabareau, N., Pham, Q., Berthoz, A., & Slotine, J. J. (2008). Where neuroscience and dynamic system theory meet autonomous robotics: A contracting basal ganglia model for action selection. *Neural Networks*, *21*(4), 628–641.

Gulwani, S. (2010). Dimensions in program synthesis. In *ACM SIGPLAN symposium on principles and practice of declarative programming* (pp. 13–24).

Guo, Q., & Zhang, M. (2010). An agent-oriented approach to resolve scheduling optimization in intelligent manufacturing. *Robotics and Computer-Integrated Manufacturing*, *26*(1), 39–45.

Hongli, L., Hongjian, W., Qing, L., & Hongfei, Y. (2016). Task allocation of multiple autonomous underwater vehicle system based on multi-objective optimization. In *IEEE international conference on mechatronics and automation* (pp. 2512–2517)

Hu, J., & Xu, Z. (2013). Brief paper: Distributed cooperative control for deployment and task allocation of unmanned aerial vehicle networks. *IET Control Theory Applications*, *7*(11), 1574–1582.

Huang, M., Narayana, V., Bakhouya, M., Gaber, J., & El-Ghazawi, T. (2012). Efficient mapping of task graphs onto reconfigurable hardware using architectural variants. *IEEE Transactions on Computers*, *61*(9), 1354–1360.

IEEE. (2016). IEEE Xplore digital library. http://ieeexplore.ieee.org/Xplore/. Accessed January 23, 2016.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Khalfi, E. M., Jamont, J. P., Mrissa, M., & Mdini, L. (2016). A RESTful task allocation mechanism for the Web of Things. In *International Conference on Computing Communication Technologies, Research, Innovation, and Vision for the Future*, pp. 73–78.

Kim, M. H., Kim, S. P., & Lee, S. (2012). Social-welfare based task allocation for multi-robot systems with resource constraints. *Computers & Industrial Engineering*, *63*(4), 994–1002.

Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical report TR/SE-0401, Software Engineering Group, Department of Computer Science, Keele University, Keele, Staffs, ST5 5BG, UK

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.

Lagoudakis, M. G., Berhault, M., Koenig, S., Keskinocak, P., & Kleywegt, A. J. (2004). Simple auctions with performance guarantees for multi-robot task allocation. In *IEEE/RSJ international conference on intelligent robots and systems* (Vol. 1, pp. 698–705).

Lee, D. H., Zaheer, S., & Kim, J. H. (2014). Ad hoc network-based task allocation with resource-aware cost generation for multirobot systems. *IEEE Transactions on Industrial Electronics*, *61*(12), 6871–6881.

Liu, L., & Shell, D. (2011). Multi-level partitioning and distribution of the assignment problem for large-scale multi-robot task allocation. In *Proceedings of robotics: science and systems*. https://doi.org/10.15607/RSS.2011.VII.026.

Liu, L., & Shell, D. (2012). A distributable and computation-flexible assignment algorithm: from local task swapping to global optimality. In *Proceedings of robotics: science and systems*.

Liu, L., & Shell, D. (2013). Optimal market-based multi-robot task allocation via strategic pricing. In *Proceedings of robotics: science and systems*.

Liu, L., & Shell, D. A. (2012). Tunable routing solutions for multi-robot navigation via the assignment problem: A 3D representation of the matching graph. In *IEEE international conference on robotics and automation*, pp. 4800–4805.

Lozenguez, G., Mouaddib, A. I., Beynier, A., Adouane, L., & Martinet, P. (2013). Simultaneous auctions for "Rendez-Vous" coordination phases in multi-robot multi-task mission. In *IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT) WI-IAT '13*, (pp. 67–74). IEEE Computer Society.

Luo, L., Chakraborty, N., & Sycara, K. (2015). Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks. *IEEE Transactions on Robotics*, *31*(1), 19–30.

Marler, R. T., & Arora, J. S. (2009). The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, *41*(6), 853–862.

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Hoboken: Wiley.

Miyata, N., Ota, J., Arai, T., & Asama, H. (2002). Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Transactions on Robotics and Automation*, *18*(5), 769–780.

Mosteo, A. R., & Montano, L. (2007). Comparative experiments on optimization criteria and algorithms for auction based multi-robot task allocation. In *IEEE international conference on robotics and automation*, pp. 3345–3350.

Nam, C., & Shell, D. A. (2015). Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Transactions on Automation Science and Engineering*, *12*(3), 889–900.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. In *International conference on principles and practice of constraint programming, CP'07*.

Nunes, E., Nanjanath, M., & Gini, M. (2012). Auctioning robotic tasks with overlapping time windows. In *International conference on autonomous agents and multiagent systems, AAMAS '12*, (pp. 1211–1212). International Foundation for Autonomous Agents and Multiagent Systems.

Okamoto, S., Brooks, N., Owens, S., Sycara, K., & Scerri, P. (2011). Allocating spatially distributed tasks in large, dynamic robot teams. In *International conference on autonomous agents and multiagent systems, AAMAS '11*, (pp. 1245–1246). International Foundation for Autonomous Agents and Multiagent Systems.

Parker, J., & Gini, M. (2014). Tasks with cost growing over time and agent reallocation delays. In *International conference on autonomous agents and multi-agent systems, AAMAS '14*, (pp. 381–388). International Foundation for Autonomous Agents and Multiagent Systems.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, *176*(2), 774–793.

Pfaff, P., Burgard, W., & Fox, D. (2006). Robust monte-carlo localization using adaptive likelihood models. In *European robotics symposium*, (pp. 181–194). Springer.

Prescott, T. J., Gonzlez, F. M. M., Gurney, K., Humphries, M. D., & Redgrave, P. (2006). A robot model of the basal ganglia: Behavior and intrinsic processing. *Neural Networks*, *19*(1), 31–61.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. (2009). ROS: An open-source robot operating system. In *ICRA workshop on open source software*.

Sikora, C. G. S., Lopes, T. C., & Magatão, L. (2017). Traveling worker assembly line (re) balancing problem: Model, reduction techniques, and real case studies. *European Journal of Operational Research*, *259*(3), 949–971.

Suemitsu, I., Izui, K., Yamada, T., Nishiwaki, S., Noda, A., & Nagatani, T. (2016). Simultaneous optimization of layout and task schedule for robotic cellular manufacturing systems. *Computers & Industrial Engineering*, *102*, 396–407.

Sugiyama, A., Sea, V., & Sugawara, T. (2016). Effective task allocation by enhancing divisional cooperation in multi-agent continuous patrolling tasks. In *International conference on tools with artificial intelligence*, pp. 33–40.

Sujit, P. B., Sinha, A., & Ghose, D. (2006). Multiple UAV task allocation using negotiation. In *International joint conference on autonomous agents and multiagent systems, AAMAS '06*, (pp. 471–478). ACM

Szomiki, S., Turek, W., Abiska, M., & Cetnarowicz, K. (2015). Multivariant planing for dynamic problems with agent-based signal modeling. *Procedia Computer Science*, *51*, 1033–1042.

The Association for Computing Machinery: ACM Digital Library. http://dl.acm.org/ (2016). Accessed January 23, 2016.

Tindell, K. W., Burns, A., & Wellings, A. J. (1992). Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems*, *4*(2), 145–165.

Tolmidis, A. T., & Petrou, L. (2013). Multi-objective optimization for dynamic task allocation in a multi-robot system. *Engineering Applications of Artificial Intelligence*, *26*(56), 1458–1468.

Wang, Z., Li, M., Li, J., Cao, J., & Wang, H. (2016). A task allocation algorithm based on market mechanism for multiple robot systems. In *IEEE international conference on real-time computing and robotics*, pp. 150–155.

White, D., & Cano, J. (2017). Task variant allocation repository. https://github.com/ipab-rad/task_alloc Accessed October 23, 2017.

Wirtschaftsinformatik. (2017). Conference ranking. http://www.wi2.fau.de/_fileuploads/research/generic/ranking/index.html Accessed January 19, 2017, http://web.archive.org.

Wolsey, L. A. (2008). *Mixed integer programming. Wiley encyclopedia of computer science and engineering*. Hoboken, NJ: John Wiley & Sons, Inc.

Zhang, Y., & Parker, L. E. (2013). Considering inter-task resource constraints in task allocation. *Autonomous Agents and Multi-Agent Systems*, *26*(3), 389–419.

Zhao, W., Meng, Q., & Chung, P. W. H. (2016). A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE Transactions on Cybernetics*, *46*(4), 902–915.

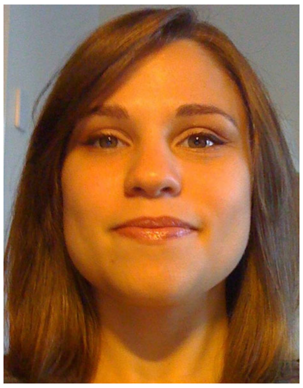
Zorbas, D., Pugliese, L. D. P., Razafindralambo, T., & Guerriero, F. (2016). Optimal drone placement and cost-efficient target coverage. *Journal of Network and Computer Applications*, *75*, 16–31.

José Cano is a research associate at the Institute for Computing Systems Architecture in the School of Informatics at University of Edinburgh since January 2014. He received his Ph.D. in Computer Science from Universitat Politècnica de València in 2012. He was a postdoctoral researcher in the Department of Computer Architecture at Universitat Politècnica de Catalunya for two years. His research interests are in the areas of computer systems, computer architecture, and systems/architectural support for robotics and deep learning. He is member of IEEE and ACM.

David R. White is a researcher in the Department of Computer Science at UCL. He has a Ph.D. in Computer Science from the University of York, where he published some of the seminal papers on both creating and improving software using heuristic search. He subsequently worked as a SICSA Research Fellow at the University of Glasgow, where he led the Raspberry Pi Cloud project, and later worked on the EPSRC AnyScale project. At UCL, he is part of the EPSRC DAASE project in automated and adaptive software engineering. His research interests include program synthesis through heuristic search and the optimisation of non-functional properties of mbedded systems.

Alejandro Bordallo received the MEng degree in Robotics and Cybertronics engineering from Heriot-Watt University, Edinburgh, UK, in 2011, and the MRes degree in Neuroinformatics and Computational Neuroscience from the University of Edinburgh, UK, in 2012. He is currently finishing the Ph.D. degree in robotics at the Robust Autonomy and Decisions group (RAD), from the University of Edinburgh, UK, in 2017. He is currently working in Robotical Ltd., an educational robotics startup, as Chief Robotics Officer in UK from 2016, and in FiveAI, an autonomous driving startup in UK from 2017. His current research interests include intention-aware motion planning, autonomous navigation, counterfactual reasoning, human–robot interaction.

Ciaran McCreesh is a Research Associate at the University of Glasgow. His primary research interests are in practical parallel algorithms, particularly in relation to hard subgraph problems. His publications cover combinatorial search, parallel algorithms, and constraint programming.



Anna Lito Michala is currently a Ph.D. Researcher at the University of Strathclyde and a Research Assistant at the University of Glasgow. She is at the third year of her studies and has also contributed to various European projects through software development. Her previous work experience as a software engineer includes development of industrial data acquisition systems, software and firmware development as well testing and verification. Her research interests include Embedded Software, Data Acquisition, Monitoring and IoT applications. She has worked across the stack of software development from drivers to web based user interface.

Jeremy Singer is a lecturer in Complex Systems Engineering. His research interests include Java runtime systems and compiler optimisations for multi-cores. He obtained his Ph.D. from Cambridge in 2006 and algorithms described in his thesis have been adopted by the widely used LLVM compiler. He has published over 30 papers covering parallelism, memory management and cloud computing. His research has been funded by Amazon and the London Mathematical Society. He is on the organising committee for major international conferences (ASPLOS, LCTES, ISMM) and a member of the UK Memory Management Network steering committee, the HiPEAC and SICSA networks.

Vijay Nagarajan is a Reader (Associate Professor) at University of Edinburgh. Dr. Nagarajan received his Ph.D. from University of California, Riverside and M.S. from University of Arizona. He is a recipient of Intel Early Career Faculty Award and a best-paper award at PACT.