Preprint of the paper which appeared in:
E. Menegatti, N. Michael, K. Berns, H. Yamaguchi (eds.):
Intelligent Autonomous Systems 13 - Proc. of the 13th International Conference IAS-13 in 2014. Advances in Intelligent Systems and Computing, Vol. 302, Springer, pp. 965-978, 2016.

# Online Interaction of a Human Supervisor with Multi-Robot Task Allocation

Karen Kurowski and Oskar von Stryk *

Simulation, Systems Optimization and Robotics Group, CS Dept.
Technische Universität Darmstadt, Germany
{k.kurowski|stryk}@sim.tu-darmstadt.de

**Abstract.** In this paper an approach is presented that allows a human supervisor to efficiently interact with task allocation in a multi-robot team (MRTA). The interaction is based on online modification of the setting of the employed MRTA optimization algorithm during its computation. For the example of a computationally expensive mixed-integer linear programming algorithm it is demonstrated how to achieve up to optimal solution quality, while simultaneously reducing the required calculation time compared to a fully autonomous optimization. The supervisor is enabled to rate feasible, intermediate solutions based on objective or subjective quality criteria and personal expertise. In that way, also suboptimal solutions can be chosen to be satisfactory, and the solver can be terminated without the need to wait for the completion of the computation of the optimal solution. An event based communication concept with queries is used as an efficient means of implementation of the interaction. Furthermore, the supervisor can support the MRTA solver in finding good solutions by defining crucial parts of the solution structure. These intuitive commands are internally translated into constraints and are added to the problem as lazy constraints. This combination of human expertise and state-of-the-art optimization algorithms allows to achieve up to potentially optimal task allocation in much shorter time.

## 1 Introduction

Multi-robot task allocation (MRTA) is a key element in mission planning and execution of autonomous robot teams. It deals with assigning a set of $m$ tasks to a group of $n$ robots in the best possible way. This problem is known to be NP-hard [7]. This means, calculating the optimal solution can take very long time in case the number of tasks is high and/or the team of robots is large. Therefore, usually heuristics are applied, to find feasible solutions in shorter time. However, in many cases these solutions are not good enough, unless the applied algorithms account for specific situations. As an example consider the setup depicted in Figure 1. Here, two quadrotors are supposed to explore a small

* This research has been supported by the German Research Foundation (DFG) within GRK 1362 “Cooperative, adaptive and responsive monitoring in mixed mode environments”.

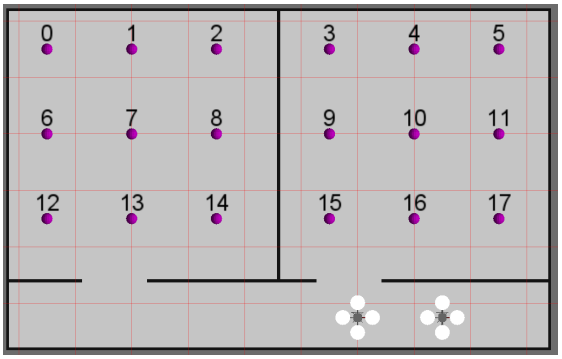


**Fig. 1.** Example scenario for joint exploration of a small indoor environment by 2 UAVs

indoor environment by visiting all of the marked locations. For a human, it is not difficult to observe that a good solution approach would be to distribute the two quadrotors to the two rooms. For a general task allocation algorithm, that is not specialized on clustering sets of tasks, this is difficult to calculate. A simple greedy scheduler for example would at first assign the tasks in the right room to the two robots, and then assign the tasks in the left room, hence both robots would work on some tasks in both rooms.

This example demonstrates, that in many cases a task allocation heuristic is not sufficient for achieving a good solution. In [15], a mixed-integer linear program (MILP) formulation for solving the MRTA problem for heterogeneous robots and arbitrary tasks with timing constraints has been presented. There interaction with a human to define crucial parts of the MILP were considered offline before the start of the MILP solver. Also premature termination of the solver by the human supervisor was not enabled. The results demonstrated that already a single input from a supervisor can significantly speed up the optimization process, but still the planning time can be very high, even for basic examples like the one in Figure 1. However, in many applications it is not necessary to calculate the optimal solution for the MRTA problem. Instead an approximate solution in between a heuristic solution and the optimum is sufficient from a practical point of view. Therefore, in this paper the approach is extended to allow a human supervisor to interact with the MRTA problem definition and solver *online* during the computation of the solutation to interactively influence the solution quality and the required computational time.

For an autonomous algorithm it is often difficult to decide if an intermediate approximation of the solution is good enough, whereas for a human it is difficult and exhausting to manually solve the full MRTA problem. Therefore, we suggest in this paper to use an autonomous algorithm for solving the MRTA problem, and have a human supervisor to interact with the solver and problem setting online during computation. The interaction concept is based on queries which are part of an event-based communication system as presented in [16]. It is demonstrated in this paper, how queries and additional MILP constraints can be used to allow

a human supervisor to interact with a state-of-the-art optimization to end up with good solutions in much shorter time.

## 2 State of Research

The different approaches to solve the MRTA problem can be classified into mainly three categories: centralized, (usually distributed) marked-based, and behavioral approaches.

If the mission can be described as an optimal assignment problem (OAP) [5], the optimal solution can be found, e. g., using the Hungarian method [11], in $O(mn^2)$ time (with $m$ robots and $n$ tasks). Search tasks can be described as a multiple traveling salesman problem (TSP, [12]), which can be transformed into a mixed-integer linear program (MILP) for faster solving [17]. The constraint optimization coordination architecture COCoA presented in [9] combines heuristic methods with a MILP formulation into an anytime algorithm, to solve complex problems with interdependencies between different goals.

Market-based approaches are usually variants of the contract net protocol (CNP) [19]. Here, the robots trade tasks for revenue, to maximize the team's overall utility. In the simplest form, this results in a greedy scheduler, like MURDOCH [6]. This solution is 3-competitive to the optimal solution, which is the best possible performance bound if neither planning in advance nor task re-allocation is allowed [8]. With the M+ architecture [2], also task re-allocation is allowed, and the robots plan one task in advance to achieve a higher solution quality. One of the first market-based approaches is TraderBots, described in [3]. In [4], this architecture is extended to robot leaders, that centrally optimize the allocation within subgroups of robots. In [18], each robot maintains a rough schedule of its future actions, and inserts traded tasks into this schedule.

In the context of behavioral approaches, each robot selects its actions based on local information. Cooperation and coordination emerge usually implicitly. First behavioral approaches were inspired by collective behavior of insects like ants and bees, without using explicit communication among the individual robots [10]. In ALLIANCE [14], the robots broadcast their current activities to their teammates. The agents are motivated to execute tasks based on impatience and acquiescence, which allows them to take over tasks from other robots. With STEAM [20], robots use shared plans and joint intentions, which enables also intentional teamwork among the robots.

None of the above approaches considers online interaction with a human supervisor to interactively influence the solution procedure. Not much work is known which considers the interactive modification of a complex task allocation algorithm by a human supervisor. For realizing the interaction, we suggest a concept based on queries and event-based communication in addition to our previous work as described next.

## 3 Queries as Interaction Mode Between Robots and a Supervisor

A communication system has been presented in [16], that includes the use of queries as robot-initiated interaction mode. The robots are enabled to transfer decisions to a human supervisor instead of taking critical decisions autonomously. The decision mode (autonomous or with human support) can be changed dynamically during runtime. An enhanced version of this concept, featuring more query modes and an improved query manager, is applied here.

A query is a specific event that can be sent occasionally by any part of a robot's software components. A query contains a description of the required decision and a set of possible solutions. This can be either a fixed set of static solutions, or can allow free inputs if appropriate, for example for numerical values. For each query, a response event is expected, containing one of the originally stated possible solutions. Additionally, queries can be tagged to different topics, and can have payloads like text or images. Policies (c. f. [16]) based on topics can be used to adapt the query mode, and hence regulate the robots' level of autonomy (LOA). This allows to trade off full autonomy for the amount of queries and resulting interruptions for each scenario independently.

To allow adapting the LOA during runtime, a query manager is used. It collects all queries that are generated by the robot's behavior control software, and determines the response, taking into account the currently granted authority of the robot.

In case a decision is transferred to the supervisor, three possible modes for presenting the queries are considered:
*Supervisor decision (SD):* All possible answers, including variable parameters, are presented, without differentiating between the options. The supervisor has to select one of the options, and if necessary specify variable parameters.
*Autonomous with confirmation (AC):* In addition to SD, a potentially best solution is determined by the query manager, and highlighted for the supervisor. The human can either confirm the preselected solution, or select one of the other available options.
*Autonomous with veto (AV):* The solutions are presented as with AC, but if the supervisor does not veto the suggested solution, this answer is given automatically.

If a decision is not transferred to the supervisor, the robots have to select a solution autonomously. Also in this case, several different modes are possible:
*Autonomous Default (AD):* One of the possible solutions is defined as default answer, which is always selected.
*Autonomous Random (AR):* One of the possible solutions is chosen randomly.
*Autonomous Algorithm (AA):* An algorithm is executed, that determines the best solution based on the current status of the robots and the environment.

With the different query modes, it is possible to model lots of decisions in the robot control software as queries. The supervisor can use policies to define the mode for each query type, or for groups based on tags. This allows to transparently switch between autonomous and supervised decisions, because the

querist always uses the interface to the query manager. The more queries have to be answered with supervisor support, the lower is the robots' LOA, and the higher is the workload of the human supervisor. When selecting the mode for the different query types, the supervisor should not be bothered with queries that can be easily answered autonomously, to avoid effects of human fatigue and complacency [13].

Within the context of this paper, queries are used to allow a MILP solver to interact with a human supervisor, to speed up the optimization process.

## 4 MILP Formulation of Multi-Robot Task Allocation with Timing Constraints

We already presented a mixed-integer linear program (MILP) formulation for the MRTA problem in [15]. This formulation can handle arbitrary tasks with optional timing constraints (i. e., earliest start time or latest time to be finished). The most important equations are given again here for reference, the complete MILP is presented in [15].

Given is a set of $n$ robots, $i \in R$, $0 \leq i < n$, and a set of $m$ tasks $j \in T$, $1 \leq j \leq m$. The final number $m$ of all tasks is not known in advance, because new tasks can arise during the mission, either automatically, or defined by the supervisor. Each robot $i \in R$ can only work on a single task $j \in T$ at a time, but can sequentially execute one task after another. For each robot $i \in R$, a cost matrix $K^i \in \mathbb{R}^{(m+1)\times m}$ is given, that defines the cost for executing task $j_2 \in T$ after finishing task $j_1 \in T$, with entries $\kappa_{ij_1j_2},\ j_1 \in T \cup 0, j_2 \in T$. Entries $\kappa_{i0j}$ describe the cost for executing task $j$ starting with the current configuration. The costs include the time to reach a destination and the time a robot needs to work at the target destination. Variables $\eta_{ij_1j_2}$ describe the time for robot $i$ to execute task $j_2$ after task $j_1$. $\eta_{ij_1j_2}$ may be equal to $\kappa_{ij_1j_2}$, but this is not necessary. A revenue $\rho_j$ is paid for completing task $j$.

Binary variables $x_{ij}$ are used to indicate that task $j$ is assigned to robot $i$. The robots can plan a fixed number of $p$ tasks in advance. Each task can be assigned to at most one robot. This leads to the following constraints:

$$\sum_{i \in R} x_{ij} \leq 1 \ \forall j \in T \tag{1}$$

$$\sum_{j \in T} x_{ij} \leq p \ \forall i \in R \tag{2}$$

To account for the order of tasks, that are assigned to the robots, variables $y_{ijk}$ are used to indicate, if task $j$ is the $k$-th task for robot $i$:

$$\sum_{k=1}^{p} y_{ijk} \geq x_{ij} \ \forall i \in R,\ \forall j \in T \tag{3}$$

To end up with a linear objective function, sequence variables $z_{ij_1j_2}$ are introduced, that indicate that task $j_2$ follows task $j_1$ in the schedule of robot $i$:

$$z_{ij_1j_2} \geq y_{ij_1k-1} + y_{ij_2k} - 1 \; \forall i \in R, \; \forall j_1 \in T \cup \{0\}, j_2 \in T, \; \forall \; 0 \leq k \leq p \quad (4)$$

Each task has parameters $t_{j_{\min}} \geq 0$ and $t_{j_{\max}} \leq \infty$, that describe the earliest and latest time the task can be accomplished without penalties. The time $\bar{t}_j$ is the calculated time when $j$ is scheduled to be completed. Variables $p_{1j}$ and $p_{2j}$ reflect if task $j$ is scheduled too early (and hence the robot has idle time), or too late, given the time constraints of all tasks.

$$p_{1j} \geq t_{j_{\min}} - \bar{t}_j \; \forall j \in T \quad (5)$$

$$p_{1j} \geq 0 \; \forall j \in T \quad (6)$$

$$p_{2j} \geq \bar{t}_j - t_{j_{\max}} \; \forall j \in T \quad (7)$$

$$p_{2j} \geq 0 \; \forall j \in T \quad (8)$$

$\bar{t}_j$ are modeled as recursive constraints based on the current schedule:

$$\bar{t}_j \geq z_{i0j} \cdot \eta_{i0j} + p_{1j} + t_{\text{now}} \; \forall i \in R, \; \forall j \in T \quad (9)$$

$$\bar{t}_{j_2} \geq (\bar{t}_{j_1} + p_{1j_2} + \eta_{ij_1j_2}) \cdot z_{ij_1j_2} \; \forall i \in R, \; \forall j_1, j_2 \in T \quad (10)$$

Please refer to [15] for a linear version of these equations.

The objective function sums up the costs for executing all scheduled tasks, and subtracts the respective revenues. Furthermore, exceeding the time constraints is penalized with factors $\alpha_1$ and $\alpha_2$. The final objective function, which has to be minimized, is defined as:

$$\sum_{i \in R} \sum_{j_1 \in T \cup \{0\}} \sum_{j_2 \in T} z_{ij_1j_2} \cdot \kappa_{ij_1j_2} - \sum_{i \in R} \sum_{j \in T} x_{ij} \rho_j + \sum_{j \in T} (p_{1j} \cdot \alpha_1 + p_{2j} \cdot \alpha_2) \quad (11)$$

The constraints matrix quickly gets very large. Both, the number of variables (the columns of the matrix) and the number of constraints (the rows of the matrix) grow linearly with the number of robots and with the planning horizon, and grow quadratically with the number of tasks. We showed in [15], that adding few, but crucial, constraints manually by the human supervisor (by defining a small part of the solution) can significantly speed up the optimization. In the described work, these constraints had to be given before the start of the optimization. In this paper, we extend this approach by enabling the supervisor to define parts of the solution while the optimization is already running. A further extension allows the solver to request input from the supervisor using queries as defined in Section 3.

## 5 Human in the Loop During the Optimization

In many cases, globally optimal task allocation is not needed, instead a solution is sufficient that is *good enough* but can be provided in reasonable time. Callback functions during the optimization allow to access the current incumbent, which denotes the best approximation of the solution the solver has found so far, and other relevant data. It is possible to terminate the optimization if desired, e. g., based on user-defined criteria. These criteria can be, e. g., after a predefined time limit is reached, or if the gap between the current incumbent and the best known performance bound is below a given threshold. However, frequently it is not possible to formulate adequate termination criteria beforehand, because they can be dependent on the current mission or on the needs of the supervisor. Therefore, the termination of an optimization should not be defined strictly before starting the calculation, but should rather be rated dynamically during the optimization. At this point, a human supervisor is put into the loop. The supervisor can fulfill two tasks for supporting the optimization: 1) rating if the current incumbent is good enough, and 2) defining parts of the solution.

### 5.1 Rating the Current Incumbent

Frequently, a human supervisor would subjectively rate a solution as (not) good enough, without being able to formally define a metric for this decision. Therefore, every time a new incumbent is found by the solver, the effect of its potential execution on the robot team is presented graphically to the supervisor. For this purpose, different types of queries (c. f. Section 3) are used.

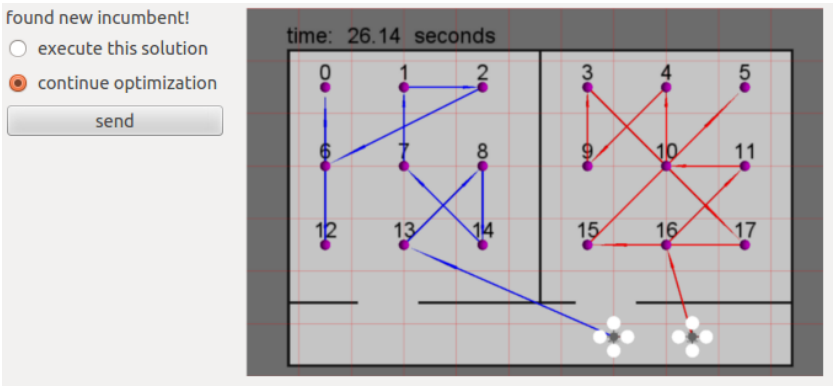


**Fig. 2.** Exemplary query dialog asking the supervisor if an incumbent is good enough.

In the simplest form, the supervisor is asked to decide if an incumbent is good enough (and hence shall be executed) or not (Figure 2). While waiting for the response, the solver can continue searching for the optimum. Therefore, this query is formulated as an AV query (autonomous decision with veto).

Besides this simple yes/no decision, the supervisor can also be asked to mark parts of a solution that are considered to be good and should be maintained for the next iteration. This is formulated as SD query (supervisor decision). In the interface, the supervisor can mark the sequences that should be maintained by clicking on them. As soon as the supervisor has finished this task, he can click the submit button to send the whole list back to the solver. Internally, additional constraints are added to the model, which require the given sequences to be in the next solution, as described in the next section. In the next iteration, these stored sequences are used as a suggestion to be kept for the next solution, which results in an AC query (autonomous with confirmation). The supervisor can either confirm to keep these sequences for the next iteration, or can change the set by adding or removing some sequences.

### 5.2 Defining Parts of the Solution

The supervisor is enabled to manually define parts of the desired solution, similar to [15]. However, while in [15] all user inputs had to be done before the start of the optimization, here the user can give input at any time. The respective constraints are added as *lazy constraints* while the optimization is running. Lazy constraints are constraints that can be added after the start of the optimization to cut off some feasible solutions. The solver guarantees that all unintended solutions can be cut off using lazy constraints.

*Assigning specific tasks and sequences to the schedule of robot $i$:* The supervisor can assign tasks $j_1 - j_k$ to the schedule of robot $i$ by selecting the respective mode in the user interface for robot $i$, and then clicking on all tasks that should be assigned to this robot. The number $k$ of assigned tasks needs to be less or equal to the planning horizon $p$, otherwise an error occurs and the supervisor is asked to deselect some tasks for robot $i$. For the MILP, the assignment of a task is translated into the constraint

$$x_{ij} = 1 \text{ for } j = 1, ..., k \tag{12}$$

These constraints do not regulate the order of the assigned tasks, it is only required that the respective tasks are somewhere in the schedule of robot $i$. To define an ordering, the supervisor can select the sequence mode for robot $i$, and mark ordered pairs to be included as sequences in the robot's schedule. This can be done without explicitly assigning the tasks first, because adding the sequences implies that the task are assigned to the robot. For a sequence $j_1 j_2$, the following constraint is generated:

$$z_{ij_1j_2} = 1 \tag{13}$$

In case a task is assigned to two robots, the supervisor receives a query to resolve this conflict.

*Executing a specific task at a higher priority:* The supervisor can request that a task $j$ is executed within the next $N$ steps in the schedule of any robot, with $N < p$. Internally, this results in the constraint

$$\sum_{i \in R} \sum_{k_1}^{N} y_{ijk} = 1 \tag{14}$$

However, it is more intuitive for the supervisor to define a time limit, which is directly assigned to $t_{j_{\max}}$. To definitely ensure the timely execution, a further constraint needs to be added:

$$\bar{t}_j \leq t_{j_{\max}} \tag{15}$$

But this can in the worst case make the problem infeasible if the deadline cannot be met by any of the robots. Therefore, this method should only be used carefully. An alternative to raise the priority for executing task $j$ is to either raise the revenue $\rho_j$, or to lower the cost $\kappa_{ikj}$ to execute this task $\forall i \in R$, $k \in T \cup 0$.

*Defining groups for joint execution:* Consider the scenario in Figure 2. If not more than two robots are available to work on these tasks, it is apparently a good solution to execute the tasks in each room as separate groups. The supervisor can enter the group definition mode, and then mark all tasks that belong to this group $J$. To model this command as a soft constraint, the costs to execute sequences within the group $J$ are lowered. Depending on how strong this soft constraint is intended to be, $\kappa_{ij_1j_2}$ is scaled with a factor $0 < a < 1$, $\forall i \in R$, $j_1, j_2 \in J$. The smaller the factor $a$ is chosen, the more likely will a robot that works on one of these tasks also execute the other tasks in group $J$. To model the same request as a hard constraint, the following constraints with new binary variables $g_i$ are added to the system:

$$k \cdot g_i \leq \sum_{j_1, j_2 \in J} z_{ij_1j_2} \, \forall i \in R \tag{16}$$

$$\sum_{i \in R} g_i = 1 \tag{17}$$

These constraints require that one robot has at least $k$ tasks of $J$ in its schedule. Parameter $k$ must not be larger than the planning horizon $p$, otherwise the problem is infeasible.

*Excluding a specific sequence from the solution:* The supervisor can exclude the consecutive execution of two tasks $j_1$ and $j_2$, for example to reduce the set of feasible solutions and hence speed up the optimization. This can be achieved by adding the following constraints:

$$z_{ij_1j_2} = 0 \, \forall i \in R \tag{18}$$

The described interaction types allow a human supervisor to support the solver in finding the optimal solution by either cutting off non-optimal solutions

quickly, or by pointing out ways to quickly find better solutions. Hard constraints lead to a high speed-up, because they require fixed values for some variables. Soft constraints lead to a larger gap between the objective value of the optimum and other solutions. Hence, also soft constraints lead to a faster calculations, but this effect is stronger with hard constraints [15]. However, hard constraints can lead to infeasible problems. Therefore, it should be decided carefully if the system should translate the input of the supervisor into soft or hard constraints.

Infeasible commands that can be detected during the input are directly rejected, if possible with a hint for the supervisor. Sometimes, the combination of different commands causes an infeasibility. In this case, it is usually difficult to automatically explain the reason to the supervisor. For handling such situations, the easiest way to resolve this problem is to reject the most recent constraints. Another option is to compute an irreducible inconsistent subsystem (ISS) that causes the infeasibility, and either remove these constraints or present them to the supervisor for further inspection. However, this requires the supervisor to have a detailed knowledge about the model, which is not assumed to be the case. Instead of removing constraints, slack variables can be used to meet the constraints as good as possible. In all cases, some commands of the supervisor cannot be addressed properly.

## 6 Application

The implementation in the presented experiments uses the robot operating system ROS (`www.ros.org`). The MILP is modeled using the python interface of Gurobi [1].

### 6.1 Rating the Current Incumbent

Consider the experiment setup of Figure 1. To find the optimal solution, the planning horizon $p$ has to be set to 9. On an Intel Core i7 with 4x 3.5GHz and 16GB RAM, this could not be calculated for memory reasons. The calculation for $p = 8$ took almost 18 hours [15]. However, when having a look at the incumbents found during the optimization for $p = 9$ (Figure 3), it can be seen that the optimal solution is actually found much faster (i. e., after 2800 seconds), while the rest of the calculation time is needed for proving that no better solution exists. Furthermore, also one of the suboptimal solutions, that are found after much shorter calculation time, can be good enough with respect to the subjective quality criteria of a human supervisor.

To leave the decision whether a solution is good enough or not to the supervisor, a query is generated every time the solver finds a new incumbent (Figure 2), as described in Section 5.1. The incumbents for this example can be seen in Figure 3. In the first three feasible solutions, the robots switch between the two rooms, and therefore these solutions are not good enough, on the one hand because execution of this solution takes too long, and on the other hand because the risk of colliding robots is very high with these solutions. Already after less

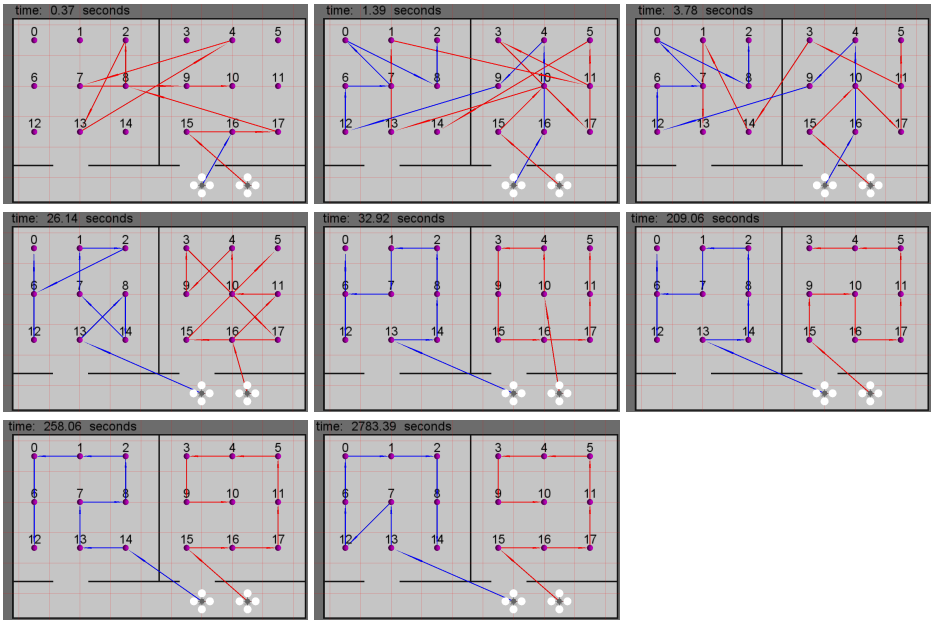


**Fig. 3.** Incumbents of the optimization for the considered example. The calculation time (measured from the start of the optimization until the incumbent is found) is displayed on top of every incumbent.

than 30 seconds, the structure of separating the two drones to the two rooms is found, but the paths of both robots are very disordered. However, only 6 seconds later a solution is found that looks much cleaner. This is more than 10 seconds faster than the average calculation time for planning horizon 4 (with fully autonomous solving), but the solution quality is much better, because none of the robots has to switch rooms. After 209 and 258 seconds of calculation time, solutions are found that are very close to the optimum. This is still twice as fast as the fully autonomously calculated solution for planning horizon 5 (the minimum planning horizon for ending up with the optimal structure). Finally, after almost 2800 seconds of calculation time, an optimal solution is found. This means, that the solver, if not interrupted, spends more than 20 times the amount of calculation time to find out that this actually is the best solution.

For most supervisors, the solution found after 33 seconds of calculation time is good enough, and hence they would decide to stop the optimization at this point. However, some other supervisor might want to wait for an even better solution, and interrupt the optimization at a later point in time. Hence, queries are a good means for enabling a human supervisor to interact with the solver in a flexible manner. This shows, that already this simple interaction mode can significantly speed up the process of finding a satisfactory solution.

### 6.2 Defining Parts of the Solution

The effectiveness of the second interaction mode, defining parts of the solution, is highly dependent on the current supervisor. While one supervisor might be able to quickly determine a critical constraint, another one may be less skilled (or less lucky), and may need more time or more interaction steps to end up with comparably good results.

Furthermore, the performance of a supervisor is dependent on the user interface, i. e., how well he understands the different possibilities to interact with the system and how fast he can actually express his commands. This is influenced by the training level of the supervisor and by the design of the user interface. However, development and evaluation of the user interface is not in the focus of this work. Therefore, no experiments for this interaction mode are presented here.

The results presented in [15] can be transferred to this work. There it had been shown that a single allocation given by a human supervisor can speed up the calculation time by an order of magnitude, while simultaneously also the quality of the solution is higher already with short planning horizons. Because the supervisor can, in contrast to the previous work, add commands while the optimization is running, he has more time to determine good interactions that can be inspired by the current incumbent, and also a higher number of commands can be added. Therefore, a well-trained supervisor is expected to have an even higher impact on the solution quality and calculation time using this interactive mode, than a supervisor who only gives a couple of commands before the start of the optimization.

## 7 Conclusion

The presented methods allows a human supervisor to interact online with a state-of-the-art optimization to achieve potentially better results for the MRTA problem in the same or shorter time needed for computation. As extension to [15], the supervisor is enabled to online rate the quality of a solution based on objective or subjective criteria and user expertise, and thereby interrupt the optimization early in case an intermediate solution candidate is satisfactory. The communication of these ratings are realized using queries, which allow the optimization to get feedback from the supervisor every time a new incumbent is found. Furthermore, the computational time for optimization can be reduced by adding lazy constraints while the optimization is running. The supervisor can formulate intuitive commands, which are internally translated into MILP constraints.

The results demonstrate that these methods are adequate for interactions between a human supervisor and an optimization approach for solving the MRTA problem. The superiority of humans over machines with respect to problem solving using conceptional thinking and user expertise is utilized. At the same time, the supervisor is *not* required to perform tasks that can be better achieved

by autonomous calculations. Furthermore, it has been demonstrated that these types of interactions can have a potentially high impact on the quality of the solution and on the time required to compute these solutions. However, the actual improvement in more complex scenarios will also be depending on the user interface and on the individual supervisor. Extensive user studies and interface evaluation were not in the focus of this paper and are subject of future work.

The approach presented in this paper can be extended to general MRTA algorithms which are characterized by high computational efforts and enable interactive monitoring of their progress and modification of the problem setting.

## References


1. Gurobi optimizer reference manual, 2014. Gurobi Optimization, Inc., http://www.gurobi.com/.
2. S. da Costa Botelho and R. Alami. M+ : a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1234–1239, 1999.
3. M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *6th Int. Conf. on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.
4. M. B. Dias and A. T. Stentz. Opportunistic optimization for market-based multirobot control. In *Proc. of the 2002 IEEE/RSJ Int.l Conf. on Intelligent Robots and Systems (IROS '02)*, volume 3, pages 2714 – 2720, 2002.
5. D. Gale. *The theory of linear economic models*. McGraw-Hill, New York, 1960.
6. B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.
7. B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Int. Journal of Robotics Research*, 23(9):939–954, 2004.
8. B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.
9. M. Koes, I. Nourbakhsh, and K. Sycara. Constraint optimization coordination architecture for search and rescue robotics. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3977–3982, 2006.
10. C. R. Kube and H. Zhang. Collective robotics: from social insects to robots. *Adapt. Behav.*, 2:189–218, September 1993.
11. H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
12. E. L. Lawler, J. K. Lenstra, A. R. Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: a guided tour of combinatorial optimization*. Wiley, 1985.
13. R. Parasuraman, R. Molloy, and I. L. Singh. Performance consequences of automation-induced "complacency". *The Int. Journal of Aviation Psychology*, 3(1):1–23, 1993.
14. L. E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.
15. K. Petersen, A. Kleiner, and O. von Stryk. Fast task-sequence allocation for heterogeneous robot teams with a human in the loop. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1648 – 1655, 2013.

16. K. Petersen and O. von Stryk. An event-based communication concept for human supervision of autonomous robot teams. *Int. Journal on Advances in Intelligent Systems*, 4(3&4):357 – 369, 2011.
17. C. Reinl and O. von Stryk. Optimal control of multi-vehicle systems under communication constraints using mixed-integer linear programming. In *Proc. of the First Int. Conf. on Robot Communication and Coordination (RoboComm)*, 2007.
18. S. Sariel, T. Balch, and N. Erdogan. Incremental multi-robot task selection for resource constrained and interrelated tasks. In *Proc. of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007.
19. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, Dec. 1980.
20. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.