2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Fast Task-Sequence Allocation for Heterogeneous Robot Teams with a Human in the Loop

Karen Petersen
Technische Universität Darmstadt
Darmstadt, Germany
petersen@sim.tu-darmstadt.de

Alexander Kleiner
Linköping University
Linköping, Sweden
alexander.kleiner@liu.se

Oskar von Stryk
Technische Universität Darmstadt
Darmstadt, Germany
stryk@sim.tu-darmstadt.de

Abstract— **Efficient task allocation with timing constraints to a team of possibly heterogeneous robots is a challenging problem with application, e. g., in search and rescue. In this paper a mixed-integer linear programming (MILP) approach is proposed for assigning heterogeneous robot teams to the simultaneous completion of sequences of tasks with specific requirements such as completion deadlines. For this purpose our approach efficiently combines the strength of state of the art mixed-integer linear programming (MILP) solvers with human expertise in mission scheduling. We experimentally show that simple and intuitive inputs by a human user have substantial impact on both computation time and quality of the solution. The presented approach can in principle be applied to quite general missions for robot teams with human supervision.**

I. Introduction

During disaster relief the efficient coordination of response teams and an appropriate allocation of available resources to mitigation tasks is indispensable [4]. Different types of tasks such as exploring locations and supplying survivors with resources within certain deadlines have to be completed by several different robot types. We consider the problem of assigning heterogeneous robot teams, i.e, robots having individual capabilities, for the simultaneous completion of sequences of tasks with specific requirements such as completion deadlines. The problem is sketched by a motivating example shown in Figure 1. Two robots have to cooperatively explore target locations in an a priori known indoor environment. The goal is to reduce the total completion time, i.e., the time needed from the start until the last target has been visited. For a human operator it is easy to see that the best solution is to assign one robot to each room since otherwise both robots would explore together one room after another, which turns out to be inefficient. However, automated task allocation needs to consider a look-ahead of at least 5 tasks into the future in order to come-up with the (intuitively) better solution. State of the art constraint solvers such as Gurobi [3] need already 7 minutes for solving this simple problem. In a more realistically and challenging setup, tasks are additionally defined by completion deadlines such as it would be the case in a rescue example where survivors need to be supplied with water and health kits in time. Scheduling with time constraints is also hard for humans, which makes the deployment of automated planning techniques indispensable.

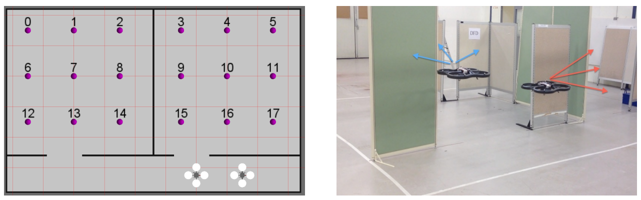

Fig. 1. Motivating example: Two UAVs are cooperatively examining targets in two separated rooms. The task-sequence assignment from UAVs to targets, i.e., each robot's set of targets and the sequence of visiting them, is computed by human-assisted optimization introduced in this paper.

Task allocation is a fundamental problem in robotics and of combinatorial complexity with increasing look-ahead. Whereas for 1-step look-ahead a polynomial centralized solution exists [6], n-step look-ahead assignments imply to solve the Traveling Salesman Problem (TSP) [8]. Constraint Satisfaction Problems (CSPs) are widely used to model combinatorial problems in AI. Solvers to these problems are either computing solutions that are satisfying constraints (classical constraint satisfaction), or optimize an objective function by selecting configurations with particular costs or utility (constraint optimization). In contrast to other works that focused on decentralized constraint optimization [14; 10], we are investing the problem of mission scheduling from a central command post [5].

In this paper we introduce a novel method for efficient human assisted task-sequence assignment. For this purpose, the strength of state of the art Mixed Integer Linear Programming (MILP) solvers is coupled with human expertise in mission scheduling. Our approach combines mission related constrains added by a human via a user interface with a constraint solver taking into account physical constraints of the environment such as travel times between different locations. We experimentally demonstrate that the proposed approach significantly outperforms automated problem solving executed on its own. Simple and intuitive inputs by a human user can have substantial impact on both computation time and quality of the solution.

The remainder of the paper is organized as follows. In Section II we discuss related work and in Section III, we provide a formal description of the problem that we are interested in. In Section IV we formulate our solution based on MILP and in Section V its extension for adding a human into the loop is presented. In Section VI we describe our experimental setup and present our results and conclude in Section VII,

II. Related Work

For solving constraint satisfaction problems, either centralized or distributed methods have been presented in the past. As already mentioned, 1-step look-ahead can be solved centrally in polynomial time by deploying the Hungarian method [6]. However, optimal n-step look-ahead assignments are intractable also for a centralized solver already for a moderate size of n, since they imply to solve the Traveling Salesman Problem (TSP) [8]. Koes and colleagues introduced a centralized n-step look-ahead solver, the so-called *COCoA* architecture for handling coordination problems by a CSP-based approach [5]. In contrast to the presented work, they did not discuss the integration of a human supervisor. From their presented results it appears that their approach could not convince in terms of scalability towards increasing number of targets, robots, and number of look-ahead steps.

Decentralized CSP solvers were mainly designed to solve 1-step assignments that are for example related to the graph coloring problem. In the literature there exists a rich set of both complete and incomplete algorithms for solving Distributed Constraint Optimization Problems (DCOPs). Well known solvers are the Distributed Stochastic Algorithm (DSA) [14], and the distributed versions of arc consistency called distributed soft arc consistency (SAC) [9]. SAC algorithms simplify a DCOP into a soft arc consistent DCOP in a distributed manner. Each agent knows only about the constraints involving its variable and must thus communicate with neighboring agents to exchange information. A classic complete DCOP search algorithm is Asynchronous Distributed Optimization (ADOPT) [10]. ADOPT uses lower and upper solution bounds in a distributed and asynchronous manner for backtracking.

There have been several extensions to the DCOP formulation proposed. Yeoh et al. extended the static DCOP formulation towards a model considering a finite sequence of static DCOPs [13]. Their approach focused on reducing computation time when solving each static DCOP in the sequence rather than modeling dependencies between subsequent DCOPs. To this end, they proposed to incrementally solve static DCOPs while reusing results from previous computations. Constraints between static DCOPs, i.e., the explicit modeling of changes over time, are not considered in their model [11]. Other related extensions include a continuous-time model where agents have deadlines to choose their assignments [12] and a model where agents can have imperfect knowledge about their environment [7].

III. Problem Description

We consider the problem of assigning n robots to m tasks over a sequence with finite horizon. Consider, for example, the situation after an earthquake where a collapsed building has to be searched for victims. Due to potential aftershocks and further collapses it might be too dangerous for human rescue personnel to enter the building. Therefore, a team R of robots is sent to explore the building, to find victims, and to supply the victims with water and first aid kits. A human supervisor H can support the team coordination of the robots from a remote location.

An a-priori map of the building is available, where H can define a set of locations L. Each location $j \in L$ shall be examined by at least one robot $i \in R$. Whenever a robot detects a victim, it is added to the set V of victims. Each victim $k \in V$ needs to be revisited after a specific time for follow-up supplies, until it is actually rescued.

To explore a location $j \in L$, a robot $i \in R$ needs to reach the Δ^L surrounding of j (usually an ellipsoid), and scan it with a sensor, that is suitable to detect human evidence (e. g., a camera). Especially $\Delta^L.z$ (the height range of the visiting area) can typically be large, which implies that j can be either explored by a UAV (without the need to land there), or by a UGV. For supplying a victim $k \in V$ with water or health kits, a robot $i \in R$ has to approach the Δ^V radius of k, with $\Delta^V < \Delta^L$. In particular, $\Delta^V.z$ is usually small, which forces a UAV to land at k. Robot i needs to stay at k for a certain time t_1^V for providing supplies to the victim.

In general, there are n robots, $i \in R,\ 0 \leq i < n$, and a set of m tasks $j \in T = L \cup V,\ 1 \leq j \leq m$. The final number m of all tasks is not known in advance, because the supervisor H can define new tasks during the mission, and victims can be detected while the robots are working on the mission. 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. The cost κ_{ij} for robot i to execute task j is defined as the expected time required to accomplish the task. In particular this includes the time to reach a destination and the time a robot has to wait until it can do something else. Additionally, a revenue ρ_j is paid for completing t_j.

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 κ_{i0j} describe the cost for executing task j starting with the current configuration (note that j was defined to be between 0 and m).

IV. MILP Formulation

In order to formulate the problem as a mixed-integer linear program, we define binary variables x_{ij}, that equal 1 if task $j \in T$ is assigned to robot $i \in R$, and zero otherwise. The robots can plan a fixed number of p tasks in advance. Each task can be assigned to exactly one robot, or being not assigned at all (which can happen if $m > p \cdot n$). 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, we introduce binary variables y_{ijk}, that indicate, if task j is the k-th task for robot i:

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

The robots start with an empty allocation, but may not explicitly be idle later in the schedule, which results in:

$$y_{i00} = 1 \ \forall i \in R \tag{4}$$

$$y_{i0k} = 0 \ \forall i \in R, \ \forall 0 < k \leq p \tag{5}$$

No robot can have more than one task in the same slot of its schedule.

$$\sum_{j \in T} y_{ijk} \leq 1 \ \forall i \in R, \ 0 \leq k \leq p \tag{6}$$

Furthermore, a robot can only have a task k in its schedule, if it also has a task $k-1$.

$$\sum_{j \in T \cup \{0\}} y_{ij(k-1)} \geq \sum_{j \in T} y_{ijk} \ \forall i \in R, \ 1 \leq k \leq p \tag{7}$$

Given these constraints, we define the objective function as the sum of all costs, that arise for executing the assigned tasks in the given order, and subtract the revenue earned for accomplishing the assigned tasks:

$$\sum_{i \in R} \sum_{j_1 \in T} \sum_{j_2 \in T} \sum_{k=1}^{p} y_{ij_1k-1} \cdot y_{ij_2k} \cdot \kappa_{ij_1j_2} - \sum_{i \in R} \sum_{j \in T} x_{ij} \rho_j \tag{8}$$

In this form, the objective function is non-linear, because of the product $y_{ij_1k-1} \cdot y_{ij_2k}$. We solve this by introducing binary variables $z_{ij_1j_2}$, that indicate that task j_2 follows task j_1 in the schedule of robot i.

$$\begin{aligned} z_{ij_1j_2} &\geq y_{ij_1k-1} + y_{ij_2k} - 1 \\ \forall i \in R, \ \forall j_1 \in T \cup \{0\}, j_2 &\in T, \ \forall \ 0 \leq k \leq p \end{aligned} \tag{9}$$

The resulting MILP with linear objective function is given by:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in R} \sum_{j_1 \in T} \sum_{j_2 \in T} z_{ij_1j_2} \cdot \kappa_{ij_1j_2} \\ & - \sum_{i \in R} \sum_{j \in T} x_{ij} \rho_j \\ \text{subject to} \quad & \text{Constraints (1) – (7) and (9)} \end{aligned} \tag{10}$$

So far, the MILP only describes the multi-agent scheduling problem without any timing constraints. However, the victim tasks have timing constraints. Each task has parameters $t_{j_{\text{min}}} \geq 0$ and $t_{j_{\text{max}}} \leq \infty$, that describe the earliest and latest time the task can be accomplished without penalties. Based on this, we model the time $\bar{t}_j$ when task j is scheduled to be completed. The matrix E with entries $\eta_{ij_1j_2}$ describes the time for each robot to execute task j_2 after executing task j_1. This matrix may be equal to the cost matrix K, but this is not required, as K will be used later for modeling inputs from a human supervisor.

We furthermore add variables p_{1j} and p_{2j}, that reflect if a task is scheduled too early (and hence the robot has idle time), or too late, given the time constraints of all tasks.

$$p_{1j} \geq t_{j_{\text{min}}} - \bar{t}_j \ \forall j \in T \tag{11}$$

$$p_{1j} \geq 0 \ \forall j \in T \tag{12}$$

$$p_{2j} \geq \bar{t}_j - t_{j_{\text{max}}} \ \forall j \in T \tag{13}$$

$$p_{2j} \geq 0 \ \forall j \in T \tag{14}$$

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

$$\bar{t}_j \geq z_{i0j} \cdot \eta_{i0j} + p_{1j} + t_{\text{now}} \ \forall i \in R, \ \forall j \in T \tag{15}$$

$$\bar{t}_{j_2} \geq (\bar{t}_{j_1} + p_{1j_2} + \eta_{ij_1j_2}) \cdot z_{ij_1j_2} \ \forall i \in R, \ \forall j_1, j_2 \in T \tag{16}$$

Equation 16 is non-linear, but because the variables $z_{ij_1j_2}$ are binary, it can be replaced by:

$$\begin{aligned} \bar{t}_{j_2} \geq \bar{t}_{j_1} + p_{1j_2} + \eta_{ij_1j_2} - M \cdot (1 - z_{ij_1j_2}) \\ \bar{t}_{j_2} \geq 0 \ \forall i \in R, \ \forall j_1, j_2 \in T \\ \text{with } M = t_{\text{now}} + \rho_{j_2} \cdot n \cdot p \end{aligned} \tag{17}$$

So far, the time is only modeled for tasks that are also scheduled. The following constraints model the best possible time for a task, that is not yet scheduled.

$$\bar{t}_{j_2} \geq (1 - \sum_{i \in R} x_{ij_2}) \cdot \min_{i \in R} \max_{j_1 \in T} ((\bar{t}_{j_1} + \eta_{ij_1j_2}) \cdot x_{ij_1}) \tag{18}$$

Also this constraint is non-linear, on the one hand because of the min and max, and on the other hand because of the product between the variables. To linearize this constraint, we introduce more variables $\hat{t}_{ij}$, that model the time when task j could be finished, if it would be executed after the end of robot i's schedule:

$$\begin{aligned} \hat{t}_{ij_2} \geq \bar{t}_{j_1} + \eta_{ij_1j_2} - M \cdot (1 - x_{ij_1}) \\ \hat{t}_{ij_2} \geq 0 \\ \text{with } M = t_{\text{now}} + \rho_{j_2} \cdot n \cdot p \end{aligned} \tag{19}$$

With this, Equation 18 is now reduced to:

$$\bar{t}_j \geq \min_{i \in R} \hat{t}_{ij} - M \cdot \sum_{i \in R} x_{ij} \ \forall j \in T \tag{20}$$

However, Equation 20 is still non-linear because of the min. To get rid of this non-linearity, we introduce binary auxiliary variables d_{ij}, that represent if $\hat{t}_{ij}$ is the minimum value for a specific task $j \in T$, and variables $\tilde{t}_j$, that represent the minimal value. With this, Equation 20 resolves to the following constraints:

$$\tilde{t}_j \leq \hat{t}_{ij} \ \forall i \in R, \ \forall j \in T \tag{21}$$

$$\begin{aligned} \tilde{t}_j \geq \hat{t}_{ij} - M(1 - d_{ij}) \ \forall i \in R, \ \forall j \in T, \\ M = t_{\text{now}} + \rho_j \cdot n \cdot p \end{aligned} \tag{22}$$

$$\sum_{i \in R} d_{ij} = 1 \ \forall j \in T \tag{23}$$

$$\bar{t}_j \geq \tilde{t}_j - M \cdot \sum_{i \in R} x_{ij} \ \forall j \in T \tag{24}$$

Exceeding the time constraints is penalized in the objective function. Idle time for waiting until a task can be started (p_{1j}) is penalized with a factor α_1. The time a task is accomplished too late (p_{2j}) is penalized with a factor α_2. Typically, $\alpha_2 >> \alpha_1$, because not meeting a task's constraints is more critical than idle time. The final objective function is defined as:

$$\sum_{i \in R} \sum_{j_1 \in T \cup \{0\}} \sum_{j_2 \in T} z_{ij_1j_2} \cdot \kappa_{ij_1j_2} - \sum_{i \in R} \sum_{j \in T} x_{ij} \rho_j + \sum_{j \in T} (p_{1j} \cdot \alpha_1 + p_{2j} \cdot \alpha_2) \tag{25}$$

Overall, 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 quadratically with the number of tasks. Because the constraints depend on the variables, the overall size of the matrix (and therefore the required calculation time) grows much faster. For a given problem, the number of robots and tasks cannot be adjusted, therefore, reducing the planning horizon is the only possibility to reduce the problem size for solving the whole problem in reasonable time.

V. Human in the Loop

Our goal is to support the automated task assignment process with inputs from a human supervisor. These inputs can have positive impacts on both solution quality and computation time. On the one hand, an expert supervisor such as a first responder can have significant implicit knowledge that is not modeled explicitly in the system. For example, the expert knows approximately the whereabouts of people in a collapsed building after a disaster. Therefore, it would make sense to focus the search on this area although this might appear initially as a suboptimal search strategy from an abstract algorithm's perspective.

On the other hand, even basic user inputs in terms of constraints can have substantial impact on the computation time of the algorithm. To solve the MILP in real-time can already for moderate problems turn out to be computationally infeasible. Interestingly, in some scenarios crucial parts of the optimal solution can be immediately apparent to a human supervisor, but need extraordinary time to be computed by a solver. For example, a human can easily identify clusters of tasks that are optimally assigned to a single robot rather than sharing them among the team which would induce additional travel costs.

Hard constraints introduced by the supervisor can make the problem infeasible. The easiest way to resolve this is to simply reject such 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. Therefore, soft constraints should be preferred over hard constraints if possible, even though hard constraints are expected to result in a larger speed up of the computation time.

We propose a user interface, that allows the supervisor to express intuitive constraints, which are then automatically translated into constraints for the MILP. For this purpose we use the graphical user interface *rviz* from the ROS (Robot Operating System) library with *interactive markers* for expressing constraints between robots and tasks. Note that this interface is not optimized for efficiency since this is not the primary focus of this work. In the following we describe 5 different modes (A-E) of adding user feedback to the MILP.

A. The expert wants the team to search a specific region either strictly at first or at higher priority

We implement strict assignments by adding hard constraints to the MILP. Given there is a single task j that the human wants to be accomplished, the following constraint will be added:

$$\sum_{i \in R} x_{ij} = 1 \tag{26}$$

So far, this does not say anything about *when* the task has to be executed. If the task has to be among the first N tasks in the schedule of a robot with $N < p$, the following constraint is added:

$$\sum_{i \in R} \sum_{k=1}^{N} y_{ijk} = 1 \tag{27}$$

If the supervisor wants instead that the task is executed within a specific time limit, the deadline $t_{j_{\max}}$ is adapted accordingly, and a hard constraint requiring the task to be finished before the deadline is added:

$$\bar{t}_j \leq t_{j_{\max}} \tag{28}$$

To prevent that the problem may get infeasible, soft constraints can be used instead. To achieve that task j is preferably part of the optimal solution either the revenue ρ_j can be raised or the cost κ_{ikj} to execute this task can be lowered $\forall i \in R,\ k \in T \cup 0$. A soft constraint on a time limit can be added by defining a deadline $t_{j_{\max}}$, but without putting a hard constraint on $\bar{t}_j$.

B. The supervisor wants to group k tasks, to be executed jointly

As an example, consider the scenario in Figure 1. If not more than two robots are available to work on these tasks, it is apparently a good solution to execute the tasks in the left room as one group, and tasks in the right room as a second group. 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 the 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:

$$k \cdot g_i \leq \sum_{j_1, j_2 \in J} z_{ij_1j_2} \ \forall i \in R \tag{29}$$

$$\sum_{i \in R} g_i = 1 \tag{30}$$

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.

C. The supervisor wants to exclude the consecutive execution of tasks j_1 and j_2

In some cases, it can be obvious for a human, that certain sequences of tasks can never be part of any good solution. In the example in Figure 1, connections between most tasks in the left and the right room are apparently not very effective. To exclude a specific sequence $j_1 j_2$ from the set of feasible solutions, the following constraint is added:

$$z_{ij_1j_2} = 0 \ \forall i \in R \tag{31}$$

As soft constraints (the sequences are still allowed, but unlikely to be selected), the costs $\kappa_{ij_1j_2}$ for executing j_2 after j_1 can be scaled by a factor $b > 1$.

D. The supervisor wants (all or some) robots to focus on a specific task type

For example, the supervisor may decide to first have a quick look at the whole environment, to collect potential victim locations, which shall be investigated after everything is explored, and not immediately. This effect can be achieved by modifying the revenue values ρ_j. Let $T_1 \subset T$ be the tasks that the robots should focus on, and $T_2 = T \backslash T_1$. To shift the robots' focus towards tasks in T_1, the revenue for these tasks is scaled by a factor $a > 1$. If the assignment of tasks in T_2 shall be highly unlikely, the revenue can be set to 0, i. e., $\rho_j = 0 \ \forall j \in T_2$. Equivalently, the costs $\kappa_{ij_1j_2}$ for executing tasks $j_2 \in T_1$ can be reduced by a factor $0 < b < 1$, and raised by a factor $c > 1$ for tasks in T_2 respectively.

In case the tasks in T_2 shall be completely excluded, a hard constraint can be added:

$$x_{ij} = 0 \ \forall i \in R, \forall j \in T_2 \tag{32}$$

If this focus shall only be defined for a subset of robots (e. g., robots of a specific type, with a specific capability, or only for a single robot), these factors and constraints are added only for the affected robots. Adapting the revenue values is not possible in that case, because they are the same for all robots.

E. A robot's autonomy shall be stopped

Stopping a robot's autonomous actions can be necessary, e. g., for allowing an operator to manually control a robot i. This implies that the robot should not automatically be assigned to a task. To forbid that any task $j \in T$ is assigned to robot i, the following constraints are added:

$$x_{ij} = 0 \ \forall j \in T \tag{33}$$

VI. Experiments and Results

All implementations in the presented experiments are utilizing ROS (`www.ros.org`). The MILP is modeled using the python interface of Gurobi [3]. We compare the performance between a fully autonomous robot team and a supervised autonomous robot team. All supervisor inputs are given prior to the start of the mission, to avoid biases due to the interface or human performance. In the following we present results from both simulated and real-world experiments. The results from simulation are separated into a homogeneous and heterogeneous robot team cooperation scenario.

A. Simulation: Homogeneous Cooperation

In the first set of experiments, two unmanned aerial vehicles (UAVs) have to cooperatively explore a small environment consisting of two rooms as shown in Figure 1. The solver works in an incremental manner by producing sequences of multi-robot team assignments each of them having maximally the length of the pre-defined planning horizon p. Hence, p can have significant influence on the solution quality, particularly if it is smaller than the length needed by the optimal solution.

In this experiment the input by the supervisor was simply to assign for the first optimization step one robot to task 13 in the left room shown in Figure 1. Any other assignment was computed by the solver.

The results are summarized in Figure 2. We ran 10 trials per configuration, except for planning horizon 9 due to memory limits. It can be seen that up to a planning horizon of 4 tasks ahead (which is less than half of the tasks per room) the traveled distance and mission time is much higher for the autonomous trials. This is because both robots first explored the right room together, and then explored the left room together. The autonomous missions with planning horizon 4 are exceptionally bad, because after both robots explored 4 targets in the right room it was optimal to continue in the left room, and hence one robot had to return to the first room afterwards. However, for a planning horizon of 5 or higher, it already takes more than 7 minutes to autonomously calculate the solution, compared to less than 90 seconds with a single input from a supervisor. Hence, with input from a supervisor, not only a better solution is found with a smaller planning horizon, also with the same planning horizon the solution is calculated in much shorter time.

Having a look at the size of the MILP, it can be seen in Table I, that the problem size is equal for both, the supervised and the autonomous trials, except for only *one constraint*, which is the one that has been added by the supervisor. As shown in Table II, this constraint is very important since it facilitates a substantial reduction of the problem size when executing the fast presolve algorithm of the MILP solver leading to the speed-up in the subsequent computation.

We also evaluated cases in which more user input is provided such as grouping targets by soft constraints to incentives that they are handled by the same robot. In the two rooms example we let the supervisor generate two groups,

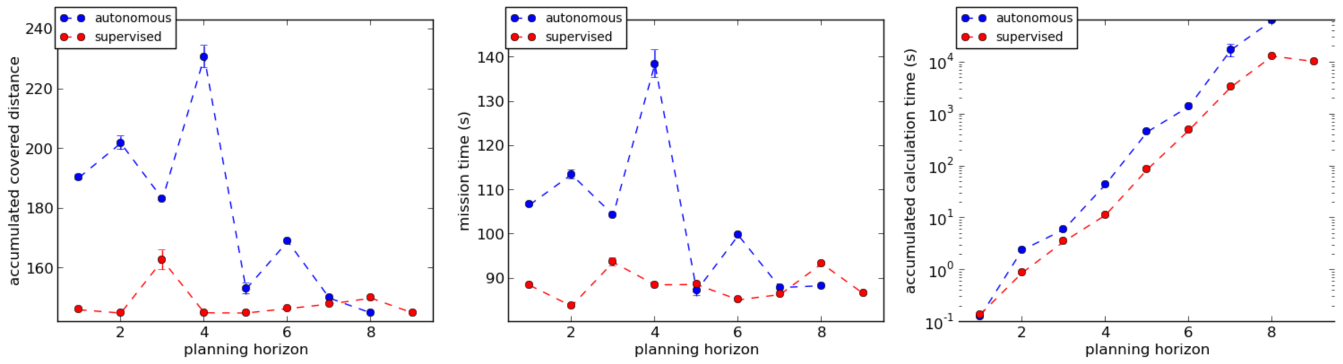

Fig. 2. Results for the two-rooms example with calculation times depicted in log scale (rightmost figure). Results for planning horizon 9 without human supervision have been omitted due to memory limits on the used computer.

one for targets in the left and one for targets in the right room. While the optimal sequence of handling tasks had still to be computed by the solver, the input significantly speeds up the computation. For planning horizon 9 the average calculation time was more than 50% lower than for the autonomous calculation with planning horizon 8. (Note that fully autonomous scheduling with planning horizon 9 could not be calculated.) The calculated optimal solution is shown in Figure 3.

planning horizon	rows		columns		nonzeroes	
	auton.	superv.	auton.	superv.	auton.	superv.
1	2262	2263	940	940	7930	7931
2	2952	2953	978	978	10132	10133
3	3642	3643	1016	1016	12334	12335
4	4332	4333	1054	1054	14536	14537
5	5022	5023	1092	1092	16738	16739
6	5712	5713	1130	1130	18940	18941
7	6402	6403	1168	1168	21142	21143
8	7092	7093	1206	1206	23344	23345
9	7782	7783	1244	1244	25546	25547

TABLE I
NUMBER OF ROWS (CONSTRAINTS), COLUMNS (VARIABLES) AND NONZEROES (DEPENDENCY BETWEEN ROWS AND COLUMNS) FOR THE PROBLEM IN FIGURE 1.

planning horizon	rows (presolved)		columns (presolved)		nonzeroes (presolved)	
	auton.	superv.	auton.	superv.	auton.	superv.
1	1064	1028	126	109	2880	2469
2	2060	1408	810	468	16704	9500
3	2712	2315	846	792	21674	21433
4	3364	3000	882	862	24048	23709
5	4016	3652	918	898	26244	26288
6	4668	4304	954	934	28458	28492
7	5320	4956	990	970	30654	30696
8	5972	5608	1026	1006	32850	32900
9	6624	6260	1062	1042	25364	23989

TABLE II
NUMBER OF ROWS, COLUMNS AND NONZEROES FOR THE PROBLEM IN FIGURE 1 AFTER APPLYING PRESOLVE.

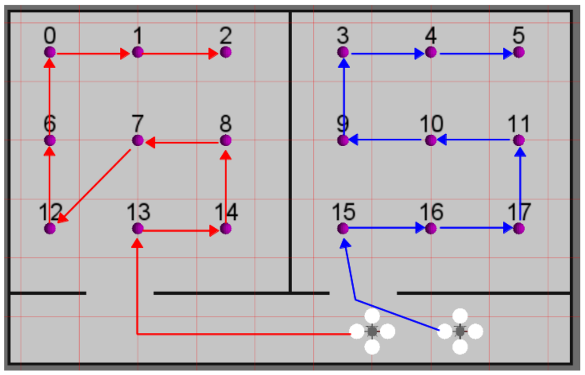

Fig. 3. Optimal solution for the problem in Figure 1 found with planning horizon 9.

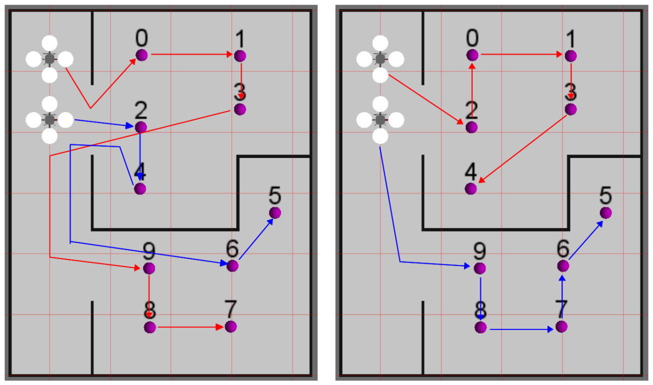

Fig. 4. Routes of the quadcopters in the real experiments. Left: fully autonomous. Right: manual allocation of task number 9.

B. Real Quadcopter Experiment: Cooperative Exploration

We used the same experimental setup as described above for conducting real-world experiments on two *Ar.Drone 2.0* quadcopters. Since we are mainly interested in the coordination algorithm and execution times, we utilized a Vicon Nexus Tracking System composed of ten cameras and reflecting markers attached to the tracked objects for obtaining highly accurate pose estimates of the robots. On top of this, we used the same ROS-based software as in the previous experiments for trajectory planning and a controller based on Engel et al. [2]. Furthermore, we had to extend the existing ROS AR.Drone driver for allowing to control several UAVs at the same time.

As can be expected due to observations from previous simulation runs, a short planning horizon lead to the strategy where both robots are entering the first room together, visiting all targets there, and then approaching the second room. When initially assigning one task in the second room to one of the robots, targets are visited more efficiently, i. e., in both rooms simultaneously. The resulting allocations for both experiments can be seen in Figure 4.

However, we learned another important lesson when executing this experiment in the real-world instead of inside the simulation environment: The wind caused by a flying quadcopter can largely influence the flight properties of another quadcopter flying in the close vicinity. This makes it much more challenging for algorithms to control the robots properly. Therefore, the computed team coordination, that typically tends to distribute the robots, is not only reducing mission time, but also lowers the risk of colliding quadcopters. The video accessible at `http://youtu.be/zojjc2FOfQA` shows the performance of the UAVs with and without supervisor support.

C. Simulation: Heterogeneous Cooperative Search and Rescue

For experiments with a heterogeneous robot team, we considered the more complex scenario of a large RoboCup rescue arena as used in RoboCup 2009 (Figure 5). The arena features different terrain types (flat, ramps, light and heavy stepfields), that require the robots to have different navigation capabilities. The three robots in the team are a wheeled robot, a tracked robot, and a UAV. The wheeled robot can navigate fast on flat floor, but cannot negotiate stairs or stepfields. The tracked robot can negotiate all obstacles except walls, but is very slow. The UAV can fly to all locations very fast, but landing is risky, especially on stepfields, or even impossible, for example on stairs or steep ramps.

Since in this scenario different robot types are inducing different travel costs with respect to the terrain, an efficient path cost planner has been utilized. We base our travel cost planner on value iteration, a popular dynamic programming algorithm frequently used for robot planning [1]. As shown in Figure 5 the planner takes as input a classified elevation map in which important structural elements such as stairs and ramps are discriminated. Value Iteration computes efficiently for each grid cell (e_x, e_y) on the elevation map the costs for reaching a goal cell (g_x, g_y). These costs are composed of travel distance as well as costs for overcoming different types of terrain indicated by the classification.

We ran three experiments with heterogeneous robots in the large arena. The results are summarized in Table III.

For the first experiment in this arena, we defined search tasks in the two flat areas (in the bottom left around the starting location, and in the upper right area) and on the second level, that is reachable via stairs or a steep ramp (Figure 6). A human can recognize quite easily, that it is best to send the UAV to the other side of the arena (tasks 1–9), and let the ground robots work on the tasks close to the starting location (tasks 10–20), because the wheeled robot cannot cross the stepfields at all, and the tracked robot needs much more time to negotiate the difficult terrain than the UAV. However, without supervisor input and with a short planning horizon, the UAV works first on tasks close to the starting area (and hence takes away tasks from the UGVs), before flying to the other side of the arena. The supervisor input requires the UAV to have task number 7 in its first schedule. With a greedy scheduler (planning horizon 1), this assignment immediately leads to an improvement of the mission time by 35% compared to the fully autonomous trial.

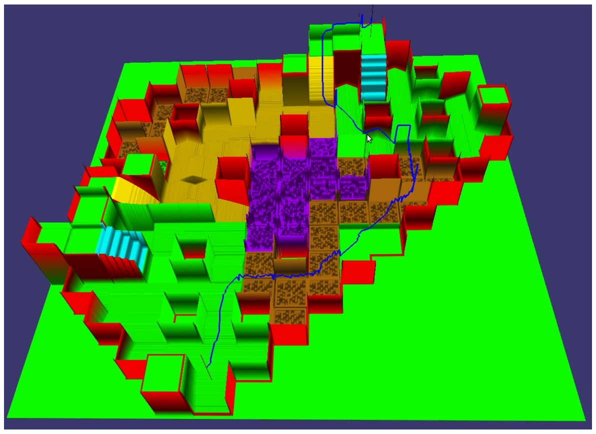
Fig. 5. Computation of plan (blue line) and plan costs on classified elevation maps (Model of the RoboCup Rescue arena 2009). Showing flat floor (green), non-traversable terrain (red), stairs (cyan), ramps (yellow), heavy stepfields (violet) and light stepfields (orange) .

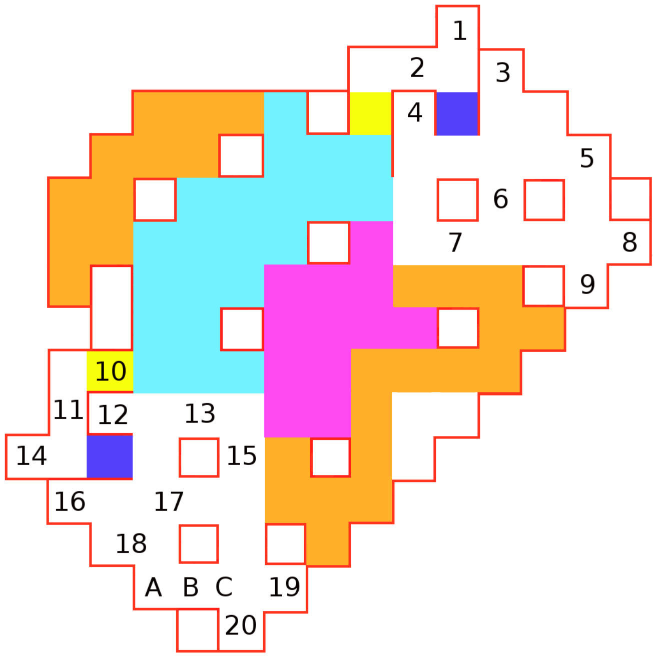

Fig. 6. Setup for experiments in the large arena. The colors specify different terrain types as in Figure 5. Numbers specify exploration tasks. A, B, and C mark the starting positions for the three robots.

In both cases, the calculation time is very low. In this specific configuration, planning horizon 4 is sufficient for the planner to autonomously send the UAV to the other side of the arena immediately. In that case, the mission time stays the same also with supervisor input, however, the time to calculate this solution is reduced by 65% to less than 1 minute with this single input. This indicates that the results from Section VI-A can be transferred to larger scenarios with heterogeneous team members.

For the second experiment, we added a timing constraint to task number 14. This constraint cannot be met by the tracked vehicle, because driving up the stair or ramp takes too much time. The wheeled robot cannot reach this location at all, hence, the only robot who can meet this constraint is the UAV. The scheduling with the timing constraints is difficult for a human, but the MILP solver can take care of this. As before, the supervisor requests the UAV to have task number 7 in its first schedule. With fully autonomous scheduling and planning horizon 2, the UAV executes a task on its way to the victim, and also executes tasks on the way to the other side of the stepfields. It turns out that the mission time for both solutions differs only marginally. However, the supervised solution required only 50% of the calculation time compared to the autonomous solution.

In the third experiment, we added a further timing constraint to task number 3. The tendency of the results are similar as before: the solution quality is similar in both trials, but the timing constraints make the MILP much more difficult to solve. The autonomous planning takes 45 seconds, while the planning with the single human input is reduced to 12 seconds.

Time constraints	**Mode**	**Planning horizon**	**Mission time**	**Travel costs**	**Calc. time**
none	autonomous	1	1160.4	2366	0.4
	supervised	1	855.8	1922	0.4
	autonomous	4	826.2	1863	159.8
	supervised	4	843.6	1874	57.7
task 14	autonomous	2	982.8	2370	9.7
	supervised	2	1009.4	2380	4.8
tasks 3, 14	autonomous	2	1096.2	5510	45.3
	supervised	2	962.6	5603	11.8

TABLE III

RESULTS FOR EXPERIMENTS WITH HETEROGENEOUS ROBOTS IN THE ROBOCUP RESCUE ARENA.

VII. CONCLUSION AND OUTLOOK

We proposed a novel approach for assigning task sequences with timing constraints to robot teams that allows to combine human capabilities in mission scheduling with state of the art constraint solving techniques. The supervisor can define intuitive constraints to the allocations, which are internally translated into constraints to the MILP. The presented experimental results clearly demonstrate the potential of this approach for both increasing the solution quality (i.e. reducing mission time), and mission assignment computation time, even when dealing with intractable problems. Besides that, automated constraint solving has the benefit of computing optimal sequences given inter-dependencies and deadlines of single tasks.

For future work, we are planning to significantly improve the user interface in order to further foster interactions between the user and the task allocation based on the MILP formulation. For large problems, interactions during the mission runtime can be expected to have a larger impact on the solution quality than interactions before the start of the mission.

ACKNOWLEDGMENT

This work is partially supported by grants from the German Research Foundation (DFG) within GRK 1362 “Cooperative, adaptive and responsive monitoring in mixed mode environments”, the Swedish Foundation for Strategic Research and the Excellence Center at Linköping and Lund in Information Technology (ELLIIT), the Research Council (VR) Linnaeus Center CADICS, the Swedish Foundation for Strategic Research CUAS Project, and the EU FP7 project SHERPA, grant agreement 600958.

REFERENCES

[1] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc. of the National Conference on Artificial Intelligence*, pages 11–18. John Wiley & Sons LTD, 1998.

[2] J. Engel, J. Sturm, and D. Cremers. Camera-based navigation of a low-cost quadrocopter. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, Oct. 2012.

[3] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2012.

[4] J.S. Jennings, G. Whelan, and W.F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proc. of the 8th Int. Conf. on Advanced Robotics, 1997. ICAR'97.*, pages 193–200. IEEE, 1997.

[5] M. Koes, I. Nourbakhsh, and K. Sycara. Constraint optimization coordination architecture for search and rescue robotics. In *Proc. of the 2006 IEEE Int. Conf. on Robotics and Automation, 2006. ICRA 2006.*, pages 3977–3982. IEEE, 2006.

[6] H.W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 2006.

[7] R.N. Lass, E.A. Sultanik, and W.C. Regli. Dynamic distributed constraint reasoning. In *Proc. of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1886–1887, 2008.

[8] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley New York, 1985.

[9] T. Matsui, M.C. Silaghi, K. Hirayama, M. Yokoo, and H. Matsuo. Directed soft arc consistency in pseudo trees. In *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1065–1072. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[10] P.J. Modi, W.M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1):149–180, 2005.

[11] F. Pecora and A. Cesta. DCOP for smart homes: A case study. *Computational Intelligence*, 23(4):395–419, 2007.

[12] A. Petcu and B. Faltings. Optimal solution stability in dynamic, distributed constraint optimization. In *Proc. of the 2007 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, pages 321–327. IEEE Computer Society, 2007.

[13] W. Yeoh, P. Varakantham, X. Sun, and S. Koenig. Incremental DCOP search algorithms for solving dynamic DCOPs. In *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, 2011.

[14] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1):55–87, 2005.