Constraint Optimization Coordination Architecture for Search and Rescue Robotics

Mary Koes, Illah Nourbakhsh, and Katia Sycara
Carnegie Mellon Robotics Institute
Pittsburgh, PA
{mberna, illah, katia}@cs.cmu.edu

***Abstract*— The dangerous and time sensitive nature of a disaster area makes it an ideal application for robotic exploration. Our long term goal is to enable humans, software agents, and autonomous robots to work together to save lives. Existing work in coordination for search and rescue does not address the variety of constraints that apply to the problem. This paper provides an expressive language for specifying system constraints. We also describe a coordination architecture capable of quickly finding an optimal or near optimal solution to the combined problems of task allocation, scheduling, and path planning subject to system constraints. We address a perceived lack of benchmarks for this research area by establishing a repository open to the research community which includes a set of benchmarks we designed to illustrate some of the complexities of the problem space. Finally, we evaluate various algorithms on these benchmarks.**

Following a natural or man made disaster, rescue workers risk their lives searching for survivors in a race against time. The search and rescue domain has the potential to benefit greatly from robotic technology. Our vision is that robots could use a rough map of the environment, perhaps obtained from a blueprint or city map, and search areas specified by human rescue workers.

This problem not only has significant humanitarian benefits but poses a difficult research challenge. Robot teams in challenging environments such as disaster sites will necessarily be heterogeneous as cost limitations, power consumption, and size constraints require tradeoffs between mobility and capabilities. Large scale disasters need to include robots with diverse modalities (e.g. ground, air, water). Multiple robots are often required to work together on a *joint goal*. Coordination on a joint goal involves simultaneously solving two $\mathcal{NP}$-hard problems, task allocation and scheduling, as well as the path planning problem [1].

Since each passing minute reduces the chance of successfully rescuing victims, the quality of the solution is very important. Furthermore, the system must accommodate a wide variety of system constraints on goals, robots, and resources. A problem formulation that does not allow for these constraints is, for example, unable to handle the simple constraint, *turn off the circuit breakers before completing any other goals.*

These four aspects of the problem–heterogeneity, joint tasks, emphasis on optimality, and additional system constraints–distinguish the problem from other multirobot planning problems. Research in market-based algorithms for coordination [2] and token based coordination algorithms [3] cannot efficiently reason about joint goals.

The contributions of this paper are fourfold. In section I, we establish the objective for the team planning problem and present a first order logic constraint language for the search and rescue domain. In section II, we present COCoA, a **C**onstraint **O**ptimization **Co**ordination **A**rchitecture, which implements the team objective and constraint language while enabling us to employ state-of-the-art optimization engines like CPLEX [4] to solve the team planning problem. Since the problem is $\mathcal{NP}$-hard, finding an optimal solution may take too long, particularly as robots must interleave planning and execution due to the uncertainty in a disaster environment. In section III, we present a progressive algorithm that finds close to optimal solutions very quickly and elegantly degrades solution quality with available time. Our fourth contribution is a set of benchmarks designed to illustrate some of the complexities of the problem space which we make available to the research community. We discuss our conclusions and future work in section V.

I. Team planning problem for search and rescue

The coordination problem for search and rescue is part of the broader class of multi-agent planning problems. Within multi-agent planning, the search and rescue problem falls into the class of *team planning* problems since robots do not have independent goals but share the team's goals. The coordination itself is *tightly coupled* in that robots must frequently work together on a *joint goal* as no single robot working on the goal is capable of achieving the goal by itself.

A search and rescue problem consists of an environment, a set of robots, $\mathcal{R}$, a set of goals, $\mathcal{G}$, and a set of additional system constraints, $\mathcal{C}$. We assume that robots have some initial representation of the environment, such as a blueprint of a collapsed building, but that there may be considerable uncertainty in the model.

The set of robots on the team is defined as $\mathcal{R} := \{R^1, R^2, ..., R^N\}$ where N is the number of robots on the team. The set of goals is $\mathcal{G} := \{G^1, G^2, ..., G^M\}$, where M is the number of goals in the system. Each goal, G^m, has an associated location in the environment, amount of time it takes to accomplish the goal or *duration*, d^m, and reward, $Q^m(t)$.

Internal to this problem representation is the assumption that there are a number of relevant capabilities; capabilities might include the ability to map the environment, check for heat signatures, manipulate objects, or extinguish fires. Each robot has a binary capability vector indicating whether or not

robot n has a relevant capability and each goal has a binary requirement vector indicating which capabilities are needed.

The last problem component is the set of system constraints, $\mathcal{C}$. The expressiveness of the constraint language for the system constraints determines the scope of problems that can be solved. Goal, robot, and resource constraints form the building blocks for this constraint language.

A. Goal constraints

Allen's 13 temporal relationships [5] (*before, equal, meets, overlaps, during, starts, finishes* and their inverses) form the basis for temporal goal constraints. Goal constraints are defined to hold if and only if both goals are scheduled, allowing for a trivial solution of not scheduling either goal. In addition to the temporal relationships, the do constraint is necessary for cases where one or more goals must be accomplished. *Do* G^m forces the goal to be scheduled at some time but does not restrict when it is scheduled.

B. Robot constraints

It is frequently convenient to specify additional constraints on the actions of individual robots. Rescue workers may wish to specify which robots should work on a goal or exclude a certain set of robots from high risk goals. This can all be effected with the *participantIn* constraint which operates on a robot and a goal. The robot constraint R^n *participantIn* G^m forces robot n to participate in goal m if the goal is scheduled.

Another important constraint is that a robot should return to base when finished with other goals. We model this with the *endAt* constraint: R^n *endAt* L means that robot n must return to location L at the end of its mission.

C. Resource constraints

Consumable resources are an important part of the search and rescue domain. Therefore, resource constraints are a necessary part of the problem formulation. We do not consider renewable resources for this domain. In general, we assume that resource constraints apply to some subteam of robots, $\mathcal{R}_s \subseteq \mathcal{R}$ and that the resource consumption for each robot is independent.

- $\mathcal{R}_s$ *limitFuelTo* F indicates that the robots in $\mathcal{R}_s$ may collectively travel no farther than F which is the amount of fuel that they possess. By setting $\mathcal{R}_s = R^n$, we can limit the amount of fuel an individual robot consumes.
- $\mathcal{R}_s$ *limitResourcesOf* $\langle f(R^n, G^m), \rho \rangle$ indicates that the robots in $\mathcal{R}_s$ may collectively consume no more than ρ of a particular resource. The $limitResourcesOf$ constraint applies to resources that are only consumed when a robot works on a goal. The function, $f(R^n, G^m)$, represents the amount of the resource robot, R^n, would consume working on goal, G^m. It must be defined for all goals in the problem, $G^m \in \mathcal{G}$, and for all robots in the subteam, $R^n \in \mathcal{R}_s$.

D. Expressive constraint language

In order to cover the full range of possible system constraints needed for search and rescue, we must be able to combine goal, robot, and resource constraints. We establish a constraint language that implements first order logic for the system constraints in $\mathcal{C}$.

DEFINITION **1.1.** A **system constraint** is an element in $\mathcal{C}$ and is denoted as $\mathbf{c}_i$. A system constraint may either be a simple constraint or a complex constraint.

DEFINITION **1.2.** A **simple constraint** is either a goal constraint (*before, equal, meets, overlaps, during, starts, finishes, do*), robot constraint (*participantIn, endAt*), or resource constraint (*limitFuelTo, limitResourcesOf*).

DEFINITION **1.3.** A **complex constraint** consists of the composition of simple and complex constraints according to the operators listed below.

Before defining the logical operators and quantifiers, we first need to explain how constraints, which we are generally defined to always be true, can be terms in boolean logic. At the top level, the general definition of a constraint still holds; the outer most clause of a complex constraint expression must always be true. However, each nested constraint is conditional upon the value of the clause above it. If this clause is true, the nested constraint must be satisfied. If this clause is false, the nested constraint must be false. The exceptions to this are resource constraints and the $endAt$ constraint which we define such that if the clause is false, the nested constraint is not required to be true. Logical operators and quantifiers include the logical not, conditional statement, if and only if, or, and, xor, forall, and exists.

II. COCoA

We have developed COCoA, a Constraint Optimization Coordination Architecture, to handle the coordination challenges posed by the search and rescue problem. The key idea behind COCoA is to preserve the semantics of the original problem by formulating it as a constraint optimization problem. COCoA uses a goal oriented representation (Figure 1) to generate a schedule that can be executed by a robot with some level of abstraction. The schedule is broken down into the necessary components of achieving a goal: traveling to the goal location, waiting for teammates who are assisting on the goal, and actually doing the goal. The maximum number of goals planned for any robot is the planning horizon, ω. Though all robots in the example in figure 1 have the same number of goals, this is not true in general; one or more robots may be idle for some or all of their planning horizon.

By modeling goal rewards as decreasing linearly with time, the problem can be modeled as a mixed integer linear programming problem (MILP) as described in [1].

A. Background MILP Problem formulation

There is the potential for confusion when discussing constraints in the MILP problem formulation. *System constraints* are defined in section I and refer to semantically meaningful

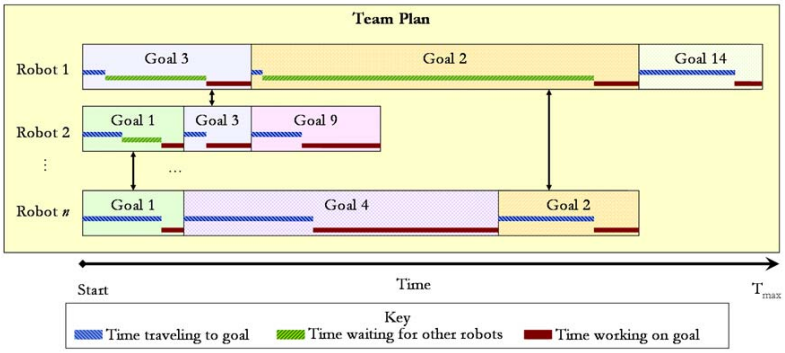

Fig. 1. COCoA uses a goal oriented representation that captures the important components of achieving a goal–traveling to the goal location, waiting for teammates who are assisting on the goal, and actually doing the goal. The maximum number of goals planned for any robot is the planning horizon, ω, ($\omega = 3$ in this example)

constraints. *Problem constraints* refer to constraints in the MILP problem formulation that implement system constraints or make the problem formulation legal.

The details of a basic problem formulation which uses both binary and real valued variables are discussed in [1]. The MILP problem formulation includes goal variables, schedule variables, and linking variables. The goal variables are of primary interest since they are used to implement the constraint language in section I.

Goals and robots are denoted with superscripts while subscripts denote a variable in the MILP problem formulation that points to the goal or robot. Therefore, G_m is the binary variable used to denote whether or not goal G^m is scheduled and R_nG_m is the binary variable that indicates whether or not robot R^n works on goal G^m. G_m_start is the real valued variable corresponding to the time at which goal G^m starts and $G_m\_start = 0$ if G_m is 0. The MILP problem formulation also has antivariables, $G_m\_start^* = G_m\_start$, if $G_m = 1$, or $G_m\_start^* = T_{max}$ if $G_m = 0$.

The real valued schedule variable, $R_nO_i\_travel$, is also important as it is used to implement the resource constraint $limitFuelTo$. It represents the time robot spends traveling on its way to its ith goal (blue line in figure 1, "Time traveling to goal").

B. Modeling constraints in COCoA

In general, goal constraints are defined to hold only if the goals are scheduled, allowing a trivial solution of not scheduling either goal. This is achieved by carefully combining goal variables and antivariables in the goal constraints. With the exception of the $before$ constraint, goal constraints are also defined so that scheduling one of the constrained goals forces the other constrained goal to be likewise scheduled. We assume that the duration, d^m, of goal G^m is greater than 0 and less than T_{max} for all goals. The simple constraints can be directly translated into problem shown in table I.

While simple constraints can be modeled with variables in the problem formulation, complex constraints require the addition of new binary constraint variables, denoted C_x.

Complex constraints can be modeled by applying the following process beginning with the outer most expression and continuing in to the nested expressions.

GOAL CONSTRAINTS	
G^x *before* G^y	$G_x\_start + d^xG_x < G_y\_start^*$ $G_x\_start^* \leq G_y\_start^*$
G^x *equal* G^y	$G_x\_start = G_y\_start$ $G_xd^x = G_yd^y$
G^x *meets* G^y	$G_x\_start + d^xG_x = G_y\_start$
G^x *overlaps* G^y	$G_x\_start < G_y\_start^*$ $G_x\_start + d^xG_x < G_y\_start^* + d^yG_y$ $G_x\_start^* + d^x > G_y\_start$ $G_x = G_y$
G^x *during* G^y	$G_x\_start^* > G_y\_start$ $G_x\_start + d^xG_x < G_y\_start^* + d^yG_y$ $G_x = G_y$
G^x *starts* G^y	$G_x\_start = G_y\_start$ $G_x\_start + d^xG_x < G_y\_start^* + d^yG_y$
G^x *finishes* G^y	$G_x\_start + d^xG_x = G_y\_start + d^yG_y$ $G_x\_start^* > G_y\_start$ $G_x = G_y$
do G^x	$G_x = 1$

ROBOT CONSTRAINTS	
R^n *participantIn* G^m	$R_nG_m = G_m$
R^n *endAt* L	We create a new goal G_{end_L} $R_nG_{end_i} = 1$

RESOURCE CONSTRAINTS	
$\mathcal{R}_s$ *limitFuelTo* F	$\sum_{n \text{ s.t. } R^n \in \mathcal{R}_s} \sum_{i<\omega} R_nO_i\_travel \leq F$
$\mathcal{R}_s$ *limitResourcesOf* $\langle f(R^n, G^m), \rho \rangle$	$\sum_{R^n \in \mathcal{R}_s} \sum_{G^m \in \mathcal{G}} f(R^n, G^m) R_nG_m \leq \rho$

TABLE I

COCoA IMPLEMENTATION OF SIMPLE CONSTRAINTS

1) If the constraint is a complex constraint introduce new binary constraint variables, denoted C_x, and constraints according to the rules for each complex expression:
   - Logical not: $\neg\mathbf{c}_1$: $C_{not} = 1 - C_1$
   - Conditional statement: $\mathbf{c}_1 \rightarrow \mathbf{c}_2$:
     $C_{if} \leq C_2 - C_1 + 1$; $C_{if} \geq \frac{C_2 - C_1 + 1}{2}$
   - IF AND ONLY IF: $\mathbf{c}_1 \leftrightarrow \mathbf{c}_2$:
     $C_{iff} \leq C_2 - C_1 + 1$; $C_{iff} \geq \frac{C_2 - C_1 + 1}{2}$;
     $C_{iff} \leq C_1 - C_2 + 1$; $C_{iff} \geq \frac{C_1 - C_2 + 1}{2}$
   - OR expression: $\mathbf{c}_1 \vee \mathbf{c}_2$
     $C_{or} \leq C_1 + C_2$; $C_{or} \geq C_1$; $C_{or} \geq C_2$
   - AND expression:$\mathbf{c}_1 \wedge \mathbf{c}_2$
     $C_{and} \geq \frac{C_1 + C_2 - 1}{2}$; $C_{and} \leq C_1$; $C_{and} \leq C_2$
   - XOR expression: $\mathbf{c}_1 \oplus \mathbf{c}_2$
     $C_{xor} \geq C_1 + C_2$; $C_{xor} \geq C_1$; $C_{xor} \geq C_2$

- FORALL quantifier: This quantifier operates with robots or goals as variables.
  We create a binary constraint variable, $C_{\forall_i}$, for each system constraint, $\mathbf{c}_{R^n}$ or $\mathbf{c}_{G^m}$. These system constraints must be formulated using this process like any other complex constraint. Additionally, we have the following problem constraint where n is the number of robots or goals: $C_\forall \geq \frac{1-n+\sum_i C_{\forall_i}}{n}$
- EXISTS quantifier: This quantifier also operates with robots as variables.
  For each system constraint, $\mathbf{c}_{R^n}$ or $\mathbf{c}_{G^m}$, we create a binary constraint variable, $C_{\exists_i}$, and require $C_\exists \geq C_{\exists_i}$, so that the *exists* constraint evaluates as true if any of the individual system constraints are true. Additionally, we add the following problem constraint so that the *exists* constraint does not evaluate to true unless one or more of the individual constraints is true: $C_\exists \leq \sum_i C_{\exists_i}$.

2) For each constraint clause, $\mathbf{c}_i$, continue to apply the rules for modeling complex expressions.
3) Simple constraints that are nested inside a complex constraint cannot be directly incorporated into the MILP as non-nested simple constraints but must be formulated as dependent upon the constraint variable, C_x, of the expression in which they are nested. Algorithm 1 describes how to generate the problem constraints so that $C_x \rightarrow \mathbf{c}_i$. Note that LHS and RHS refer to the Left (or Right) Hand Side of the inequality.

Algorithm 1 Generate problem constraints for $C_x \rightarrow \mathbf{c}_i$

Use the constraints from table I that correspond to $\mathbf{c}_i$.
Turn all equalities, $x = y$, into inequalities: $x \leq y$ and $x \geq y$.
Rearrange each inequality so that the RHS = 0
if LHS ≤ 0 or LHS < 0 **then**
  RHS = $(1 - C_x)T_{max}$
else
  RHS = $(C_x - 1)T_{max}$
end if

4) Force $\neg C_x \rightarrow \neg \mathbf{c}_i$ by applying DeMorgan's law ($\neg(A \wedge B) = \neg A \vee \neg B$) and following algorithm 2.
5) Set the top level constraint variable $C_x = 1$.

We can quickly verify that the process for establishing $C_x \rightarrow \mathbf{c}_i$ is valid: if $C_x = 0$, we have the left hand side of the expression, LHS $\{>, \geq\} - T_{max}$ or LHS $\{<, \leq\} T_{max}$. Since all times are bounded by T_{max}, this is a trivial constraint. On the other hand, if $C_x = 1$, the right hand side is 0 which is simply a restatement of the original constraint. The check for $\neg C_x \rightarrow \neg \mathbf{c}_i$ is similar, though slightly longer.

C. Solving the MILP with CPLEX

By modeling the problem as an MILP as described in section I, we can leverage a vast amount of work in solving these problems including the commercially available solver, CPLEX [4]. The standard approach for solving an MILP is to relax the constraint that certain variables be integer and solve the resulting linear program (LP) using LP algorithms such as the simplex or dual simplex algorithm. This noninteger solution is known as the *relaxed solution* and provides an upper bound on system performance. In the case that the relaxed solution is integral, this optimal solution is returned.

Algorithm 2 Generate problem constraints for $\neg C_x \rightarrow \neg \mathbf{c}_i$

Use the constraints from table I that correspond to $\mathbf{c}_i$.
Turn all equalities, $x = y$, into inequalities: $x \leq y$ and $x \geq y$.
Rearrange each inequality so that the RHS = 0
for all resulting inequality problem constraints **do**
  Create a new binary variable C_{x_i}.
  if LHS ≤ 0 or LHS < 0 **then**
    RHS = $(1 - C_{x_i})T_{max}$
  else
    RHS = $(C_{x_i} - 1)T_{max}$
  end if
end for
Add problem constraint: $\sum_i C_{x_i} + \sum_{G_m \in \mathbf{c}_i} (1 - G_m) \geq (1 - C_x)$

Since the problem is $\mathcal{NP}$-hard, generally the system must employ the branch and bound algorithm to search for the optimal integer solution. Without additional information, the search for the integer solution begins near the relaxed solution. Depending on the structure of the search space, finding a feasible solution can take a long time (hours to find a feasible solution to a large search and rescue problem and days or weeks to find the optimal solution). This is unacceptable for the search and rescue domain since planning delays the onset of execution.

III. REAL TIME ALGORITHMS

Particularly for problems with few or no constraints, it is possible to use heuristics to find solutions in polynomial time. In fact, the search and rescue problem has elements, specifically rewards that decrease over time, that suggest that greedy heuristics would perform reasonably well. Conveniently, we can combine heuristics and MILP solution techniques resulting in an algorithm superior to either individual approach.

As described in the previous section, solving an MILP involves branch and bound search. By default, this search begins around the relaxed solution. It is possible to generate an initial solution using a heuristic and use this as the starting point for the search. This enables the system to find a feasible solution quickly and then improve the solution with available time. Furthermore, the bounds in the branch and bound algorithm provide error bounds on a given solution so that it is possible to stop when the solution is within some bounds of the optimal or estimate distance of an arbitrary solution from optimal. We exploit this symbiosis between heuristic and optimal solver in our anytime algorithm (Algorithm 3).

A. Anytime Algorithm

The key idea in the anytime algorithm is to make the interaction between the optimization algorithm (CPLEX) and heuristic (any number of heuristics can be used–we describe two below) an iterative process by starting with a small subproblem and expanding this until it includes the entire

Algorithm 3 Anytime Scheduling using MILP Solver

1: $oldSoln$ = {}
2: **for** $\omega = 1$ to # Goals **do**
3: $milp \Leftarrow$ CreateMILP(ω)
4: $start \Leftarrow$ HeuristicScheduler($oldSoln$, ω)
5: $optimalSolnForPH \Leftarrow$ Optimize($milp$, $start$)
6: $oldSoln = optimalSolnForPH$
7: **end for**
8: return $optimalSolnForPH$

Algorithm 4 Myopic heuristic

1: ω = 1; $unsatGoals = Goals$; $currSchedule = \{\}$
2: **while** ($\omega \leq$ # Goals) && $!isEmpty(unsatGoals)$ **do**
3: $prob \Leftarrow$ CreateMILP(1, $unsatGoals$)
4: $partialSoln \Leftarrow$ Optimize($prob$)
5: $currSchedule \Leftarrow$ Schedule($partialSoln$)
6: $unsatGoals \Leftarrow unsatGoals-$satGoals($currSchedule$)
7: $\omega = \omega + 1$
8: **end while**
9: return $currSchedule$

problem. At each iteration, a heuristic is used to generate a solution for the subproblem. This heuristic solution is used as a starting point for search in the optimization step. Since the resulting optimized solution to this subproblem is at least as good as the heuristic solution to the same subproblem, the heuristic uses this optimized solution as its starting point and only finds a solution for the expanded portion of the problem.

B. Heuristics

Since goal rewards decrease over time, it is reasonable to expect greedy heuristics to perform well. Two greedy heuristics arise naturally from the problem variables–robots and goals. In the first heuristic, which we call the *myopic* heuristic, robots recursively schedule goals over an increasing planning horizon. The heuristic attempts to find the best solution for a planning horizon of 1. We use our MILP problem formulation and CPLEX to find this solution as shown in algorithm 4. Empirically, this is very fast–well under a second–even for large problems. This is because problem size grows exponentially with the length of the planning horizon so this subproblem is much smaller than the original problem.

The second heuristic, or *greedy goal* heuristic, is inspired by market based task allocation algorithms which have been shown to work well in a variety of domains [2]. Since the goals are known in advance, they are sorted and auctioned off in decreasing order of reward. For each required capability of each goal, robots bid their costs to provide that capability based on their current schedules. The lowest bidder is assigned to that goal capability. The greedy goal heuristic is less sophisticated than many market based systems as it does not allow robots to reauction their goals.

C. Myopic heuristic limitations

Each of these heuristics has its limitations and can lead to inefficient behavior if not combined with an optimization algorithm. The myopic heuristic performs poorly if it is best but not required for one robot to assume more goals than its teammates. For example, if there are two goals with the same requirements in the same room and two identical robots, one already in the room and one in a different building, the myopic heuristic will assign one goal to each robot although the optimal solution is clearly to assign both goals to the close robot.

D. Greedy goal heuristic limitations

The greedy goal heuristic but may perform poorly if the problem requires careful resource management. Consider a disaster site with two flooded rooms and a dry room and a team of two robots. However, only robot 1 is waterproof and able to explore the flooded rooms. If robot 1 starts slightly closer to the dry room and the goal of exploring the dry room is allocated first, the greedy goal heuristic will assign all three goals to robot 1 although it would be more efficient to assign the dry room to robot 2 to explore.

IV. Benchmarks and Results

The lack of benchmarks for heterogeneous multirobot coordination with joint tasks is an obstacle to rigorously comparing algorithms and heuristics. We have developed a collection of benchmarks designed to illustrate some of the challenges of a complex domain such as search and rescue including the limitations of the heuristics as described in the previous section. As a service to the research community, we have set up a repository to allow researchers to access our benchmarks and contribute their own: *http://www.cs.cmu.edu/~mberna/research/*.

Although the system constraints from section I add an important degree of expressiveness to the system, a baseline analysis without constraints must first be performed to better understand the problem space. Therefore, although we are able to automatically generate benchmarks with constraints which are included in the repository, this analysis limits itself to problems without additional system constraints. This also allows us to compare algorithms and heuristics that lack the expressiveness to represent and reason about these constraints.

A. Benchmarks

We implemented a program to automatically generate random environments according to certain specification which we used to generate 5 environments with 3 and 15 robots and 5 and 15 goals for a total of 20 problems in each of the following classes of benchmarks. All the benchmarks have three relevant capabilities but the profiles of robot capabilities and goal requirements distinguish the various benchmarks. The benchmark classes are described in table II. In order to limit the variations between benchmarks, all environments are randomly generated 10x10 grids with randomly assigned 4 point connectivity between nodes.

B. Results

We compared five solution techniques on these benchmarks: the myopic heuristic (Algorithm 4), the greedy goal heuristic described in section III, the anytime algorithm (Algorithm 3) with the myopic heuristic as the heuristic solver on line 4

Benchmark	Robot Capabilities	Goal Requirements	Goal Location
Homogeneous	All	All	Random
Tight	Single	All	Random
Easy Clustered	All	All	Clustered
Difficult Clustered	Single	All	Clustered
Precious Resources	Super and weak robots	Easy and hard goals	Random
Random	Random	Random	Random

TABLE II

BENCHMARK CHARACTERISTICS (SUPER ROBOTS CAN DO ANY GOAL WHILE WEAK ROBOTS ARE ONLY ABLE TO DO EASY GOALS)

of Algorithm 3, the anytime algorithm with the greedy goal heuristic as the heuristic solver, and the anytime algorithm with the "best" heuristic in which the heuristic solver compares the myopic and greedy heuristics and returns the solution with the highest utility. The optimization step (line 5 of Algorithm 3) was limited to 90 seconds. We compared the results for three different levels of time pressure, $T_{max} = 100$, 1000, and 2000. The results for T_{max} = 100 are shown in figure 2.

C. Analysis

In general, we observed that the greedy goal heuristic performed well on the homogeneous and clustered benchmarks. This matches what we know about these benchmarks and the workings of the greedy goal heuristic. The homogeneous and easy clustered benchmarks have no joint goals so the problem reduces to the optimal assignment problem on which greedy market task allocation has been demonstrated to perform well [2]. The greedy goal heuristic performs the worst on the precious resource benchmarks which matches our analysis of the limitations of the greedy goal heuristic in section III-D.

The myopic heuristic performed at least as well as the greedy goal heuristic on all benchmarks except the easy clustered benchmarks on which it performed slightly worse. This matches our analysis of the limitations of the myopic heuristic (section III-C). The performance is only slightly worse because the tasks are clustered together so that even if the tasks are allocated suboptimally for a planning horizon of 1, this is outweighed by the goals accomplished over the remainder of the planning horizon. In this sense, the clustered goal benchmarks were less successful than we had hoped in illustrating the limitations of the myopic heuristic.

The anytime algorithm was able to significantly improve performance on a number of benchmarks with a maximum improvement of over 50%. This indicates that the heuristics benefit significantly from the relatively small amount of time spent in optimization. Without using the heuristic starting solution, CPLEX is often unable to find a feasible solution in 90 seconds which illustrates the benefits of the heuristic starting solution to the MILP solver.

We found that the results from the other values of T_{max} are nearly identical to figure 2 (and so are not included here) except in scale. The utility gains when $T_{max} = 2000$ are half the utility gains for $T_{max} = 1000$. Since utility is defined as a linear function of T_{max}, we would expect a linear relationship between T_{max} and utility gain. However, the utility gains when $T_{max} = 100$ are 15 times the utility gains of $T_{max} = 1000$.

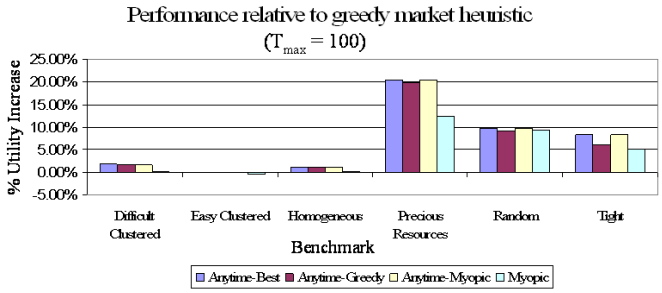

Fig. 2. Benchmark results using greedy goal heuristic as baseline averaged over 3 and 15 robots and 5 and 15 goals.

The reason for this is that as T_{max} decreases, suboptimal solutions are not able to accomplish as many goals which magnifies differences in solution quality. This illustrates the importance of optimality in the search and rescue domain where there is often significant time pressure.

V. CONCLUSIONS AND FUTURE WORK

In this paper we presented a language for modeling constraints in first order logic, an architecture based on well known mixed integer linear programming work capable of finding optimal team plans subject to these constraints, and an anytime algorithm and various heuristics to solve these problems quickly. We evaluated these solution techniques on a set of multirobot team planning benchmarks. The benchmarks analyzed here are only a beginning. We hope other researchers will contribute test cases and will continue to add benchmarks to the repository ourselves including test cases with semantically meaningful constraints. Finding optimal or near optimal team plans in the presence of system constraints is an open problem that requires the development of new algorithms. Although this paper was motivated by the search and rescue domain, the general problem formulation and solution techniques discussed are applicable to a wide range of domains such as integrated manufacturing, reconnaissance, or space exploration.

ACKNOWLEDGEMENTS

This work is supported by NSF Award IIS-0205526. The authors would also like to thank the ILOG corporation for their generous support.

REFERENCES

[1] M. Koes, I. R. Nourbakhsh, and K. P. Sycara, "Heterogeneous multirobot coordination with spatial and temporal constraints," in *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*. AAAI Press AAAI Press / The MIT Press, 2005, pp. 1292–1297.

[2] M. B. Dias, R. M. Zlot, N. Kalra, and A. T. Stentz, "Market-based multirobot coordination: A survey and analysis," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-13, April 2005.

[3] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara, "An integrated token-based algorithm for scalable coordination," in *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2005, pp. 407–414.

[4] *ILOG CPLEX 9.0 User's Manual*, ILOG, 2004.

[5] J. F. Allen, "Maintaining knowledge about temporal intervals." *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.