xBots: An Approach to Generating and Executing Optimal Multi-Robot Plans with Cross-Schedule Dependencies

G. Ayorkor Korsah, Balajee Kannan, Brett Browning, Anthony Stentz and M. Bernardine Dias

***Abstract*— In this paper, we present an approach to bounded optimal planning and flexible execution for a robot team performing a set of spatially distributed tasks related by temporal ordering constraints such as precedence or synchronization. Furthermore, the manner in which the temporal constraints are satisfied impacts the overall utility of the team, due to the existence of both routing and delay costs. We present a bounded optimal offline planner for task allocation and scheduling in the presence of such cross-schedule dependencies, and a flexible, distributed online plan execution strategy. The integrated system performs task allocation and scheduling, executes the plans smoothly in the face of real-world variations in operation speed and task execution time, and ensures graceful degradation in the event of task failure. We demonstrate the capabilities of our approach on a team of three pioneer robots operating in an indoor environment. Experimental results demonstrate that the approach is effective for constrained planning and execution in the face of real-world variations.**

I. Introduction

Multi-robot teams will increasingly be used to address heterogeneous spatially distributed tasks in domains where no single team member can effectively execute all tasks. In some cases, tasks are distributed among robots for independent execution, whereas others require constant and tightly-coupled interaction amongst robots. The multi-robot coordination problems addressed in this paper involve spatially distributed tasks related by temporal constraints. Furthermore, there may be costs associated with delays needed to ensure that temporal constraints are satisfied. Such problems have *cross-schedule dependencies* [1] because the schedules of different robots are interdependent. This rich category of problems spans domains such as emergency assistance, agriculture, and planetary exploration.

This paper addresses bounded optimal task allocation and scheduling for problems with such temporal cross-schedule dependencies. It further addresses the flexible execution of the resulting temporally constrained plans in the presence of execution-time operating variations.

The contributions of this paper are three-fold:

1) xTeam: a planner that computes an anytime, bounded optimal solution to a task allocation and scheduling problem with cross-schedule dependencies. Due to space limitations, we focus on its support of key temporal cross-schedule dependencies and omit several other details [1].
2) An approach to flexible execution of temporally constrained plans, based on the plays paradigm [2]. The execution strategy ensures that temporal constraints are satisfied despite timing variations during execution.
3) xBots: an integrated planning and execution system that combines the xTeam planner and the plays paradigm with a tool for automatically translating the computed constrained plans into a form suitable for flexible execution. We evaluate the developed approach on a a team of indoor robots performing a set of spatially distributed tasks.

The organization of the remainder of the paper is as follows. Section II presents related work while Section III describes details of the xBots approach. Results on the functionality of the planner are summarized in Section IV, while Section V describes the experimental setup and robot test results demonstrating flexible plan execution. Finally, Section VI concludes with a discussion of future work.

II. Related Work

For efficient planning, market-based strategies have been proven useful in many multi-robot task allocation problems [3], [4]. In these approaches, agents are designed as self-interested entities that operate in a virtual economy by bidding on tasks. Recent work has begun to address some inter-task dependencies [5] and also to give some bounds on worst-case performance [6]. However, market-based approaches in general do not provide optimality guarantees, particularly for time-extended task allocation.

Mathematical programming approaches express the coordination problem as a mixed integer programming problem (MIP), which can then be solved optimally. Such approaches are common in Operations Research, particularly for vehicle routing problems (VRPs). Some recent work in optimal approaches to multi-robot coordination and vehicle routing incorporates inter-task temporal constraints [7], [8]. However, this work does not plan for cases where satisfying these constraints has an impact on the overall team cost, neither does it address execution of the constrained plans, given real-world timing variations.

Popular approaches to plan execution use negotiation to formulate team plans and ensure conflict-free execution. For example, Alami et al. [9] present a framework for multi-robot cooperation to achieve independent goals (i.e. goals not related by constraints). Goals are allocated to robots using

This work is sponsored in part by the Qatar National Research Fund under contract NPRP 1-7-7-5, and the Boeing company under contract CMU-BA-GTA-1. The content of this paper does not necessarily reflect the position or policy of the sponsors and no official endorsement should be inferred.

G. Ayorkor Korsah is with Ashesi University College, Berekuso, Ghana. *Email:* ayorkor@alumni.cmu.edu. Balajee Kannan, Brett Browning, Anthony Stentz and M. Bernardine Dias are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. *Email:*{bkannan,brettb,axs,mbdias}@ri.cmu.edu

the market-based M+NTA (Negotiation for Task Achievement) scheme. The M+CTA (Cooperative Task Achievement) scheme then addresses resource conflicts by inserting temporal order constraints between actions of different robots. As another example, Joyeux et al [10] present a shared "plan database" for building, negotiating, and executing multi-robot plans. Other relevant work includes CAMPOUT by Pirjanian et al [11], a distributed multi-robot control architecture that represents joint team activities using a finite state machine augmented with synchronization primitives for tight coordination of group activities. None of these approaches address optimal planning and flexible execution of tasks related by temporal constraints.

Simple Temporal Networks (STNs) [12] represent flexible time plans by specifying task start times as windows and determining precise times only during execution. STNs have been used to enable flexible scheduling of the plans of individual agents operating as part of a team [13]. While STNs represent flexible plans, they do not by themselves represent optimal plans. The use of STNs is complementary to our approach which ensures satisfaction of cross-schedule constraints during execution of a pre-computed optimal plan.

In summary, while there is diverse literature on multi-robot coordination, a review reveals a dearth of discussion on optimal planning for problems with cross-schedule dependencies, and on multi-robot execution of constrained optimal plans.

III. Approach

The xBots system architecture consists of a planning module and an execution module, illustrated in Figure 1. The planning module computes an anytime bounded optimal allocation of tasks to agents and a schedule for task execution. That is, it computes progressively better solutions, terminating with either the optimal solution or the best solution it can find in the allotted planning time, along with bounds on suboptimality. The computed schedule specifies task start times and waiting times needed to ensure that temporal constraints are satisfied. This plan is then transferred to the execution module. In an ideal world, the computed plans could be executed as-is by the individual agents and would collectively satisfy all the cross-schedule constraints. In reality however, rigid execution strategies are susceptible to failure due to variations in the operating domain. Thus, agents must have an awareness of the high-level constraints impinging on their part of the team plan and a strategy to enable execution of the plan. The agents do not, however, need to be tightly coupled and can operate largely independently except when cross-schedule constraints need to be satisfied. To achieve this, we implement a distributed plan execution strategy, based on the *plays* [2] paradigm, in which agents are loosely-coupled and intermittently synchronize with each other. We also outline an automated approach, implemented by a "Translator" sub-module, for converting the generated multi-robot plan to a flexible plan that can accommodate real-world execution-time variations. Thus, the execution strategy can be decoupled from the xTeam planner and optionally used with other solvers.

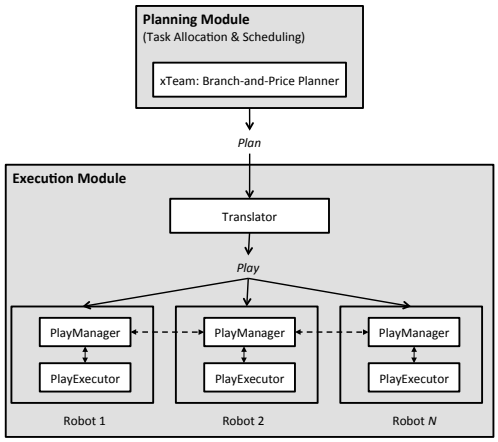


Fig. 1. xBots approach for optimal planning and flexible execution

A. Problem Formulation

A set of mobile agents, K, is available to perform a collection of tasks. Each multi-agent task can be decomposed into simpler single-agent tasks. Each single-agent task $j \in J$ requires specific agent capabilities and consists of one or more spatially distributed subtasks, $i \in I$ which may each have agent capacity requirements. Subtasks of different tasks may be related by temporal constraints, thus creating dependencies between different agents' schedules.

To enable computation of an optimal solution, we formulate a set-partitioning mixed-integer programming model, with side constraints, for this problem. The terms in this model are summarized in Table I, while the model itself appears in Figure 2.

A binary variable, x_k^r represents whether a given agent k is assigned to a particular *route* (single-agent plan), r, out of all feasible routes R_k for that agent. The real-valued execution-delay variable d_i^k represents the amount of time that agent k, having arrived at the chosen location for subtask i, has to wait before it can begin execution of subtask i. t_i represents the time that execution begins on subtask i. In addition to these domain variables, a "helper" variable, $a_{i'i}$, represents the delay in the arrival time for subtask i due to the execution-delay time for subtask i' occurring earlier on the same route. These arrival-delay variables are needed simply to ensure a linear formulation; without them, the model would need to be non-linear, containing product terms of the form $d_i^k x_r^k$.

Solving the model involves generating feasible routes and assigning values of 0 or 1 to route variables to maximize the difference between task rewards and travel and delay costs (Eq. 1 in Figure 2). *Problem constraints* in the model define the feasibility of the basic task allocation and scheduling problem. They specify that each agent must perform at most one route (C1) and each task is performed on at most one route (C2). They also ensure the consistency of the time and delay variables (C3-C5c). *Domain constraints* model the domain-specific temporal constraints, namely time window constraints (C6a-C6b), precedence constraints (C7a-C7b), and synchronization constraints (C8a-C8b).

TABLE I

DEFINED VARIABLES AND TERMS

Var.	Definition	Type
Domain Variables		
x_r^k	Whether agent k performs route r	Binary
d_i^k	Delay time of agent k for subtask i	Real
t_i	Execution start time for subtask i	Real
"Helper" Variables		
$a_{i'i}$	Arrival delay for subtask i due to subtask i'	Real
Term	**Definition**	**Type**
R_k	Feasible routes for agent $k \in K$	Set
P	Pairwise precedence constraints	Set
S	Pairwise synchronization constraints	Set
v_j	Value of completing task j.	Real
c_{1r}^k	Travel cost for route $r \in R_k$	Real
c_2^k	Wait cost per unit time for agent k	Real
π_{jr}^k	Whether task/subtask j occurs on route r	Binary
λ_i	Service duration for subtask i	Real
$\delta_{i'ir}^k$	Whether subtask i' comes before i on route r	Binary
$[\alpha_i, \beta_i]$	Valid time window for starting subtask i	Real
τ_{ir}^k	Time that agent k would arrive at subtask i on route $r \in R_k$ if no delays were necessary	Real
μ_j	Max. allowed time span for task j	Real
D_i	Max. allowed delay time for subtask i	Real
τ_∞	End of planning horizon	Real
$\epsilon_{i_1 i_2}^P$	Min. time gap between completing subtask i_1 and commencing subtask i_2 for $(i_1, i_2) \in P$	Real
$\epsilon_{i_1 i_2}^S$	Exact time gap between commencing service on subtasks i_1 and i_2 for $(i_1, i_2) \in S$	Real
y_j	Whether task/subtask j is performed $= \sum_{k \in K} \sum_{r \in R_k} \pi_{jr}^k x_r^k$	Binary

B. Plan generation

Mixed integer programming problems are generally solved in a branch-and-bound framework. To begin, a bound on the solution is computed by relaxing the integrality constraints and solving the resulting linear program. Subsequently, branching decisions are made on fractional variables that should be integer, and the solution process is repeated at each node of the branch-and-bound tree, until a solution is found that satisfies the integer constraints and whose objective value is at least as good as the best bound on the tree nodes.

In a set-partitioning model like ours, with a large number of route variables, it is not possible to enumerate all possible variables/columns in the problem up front, and this is where a *column generation* process is useful. That is, the algorithm initially considers a subset of columns (in our case, feasible routes) in the "master" problem given by the set-partioning model. Then, additional columns to be added are determined by solving a *pricing subproblem*, derived from the dual variables of the model. A *branch-and-price* algorithm [14] is a branch-and-bound algorithm in which column generation occurs at each node of the branch-and-bound tree.

xTeam is a custom branch-and-price algorithm that computes progressively better solutions, with bounds on quality, until it returns a provably optimal solution. Due to space limitations, we omit its full details [1]. However, we briefly describe the pricing subproblem used for column generation, and the branching decisions used to ensure feasible solutions.

Designating the dual variables of our model as u_b^a (where

Maximize:

$$\sum_{j \in J} \sum_{k \in K} \sum_{r \in R_k} v_j \pi_{jr}^k x_r^k - \sum_{k \in K} \sum_{r \in R_k} c_{1r}^k x_r^k - \sum_{i \in I} \sum_{k \in K} c_2^k d_i^k \tag{1}$$

Subject to: *Problem Constraints:*

$$\sum_{r \in R_k} x_r^k \leq 1 \qquad \forall k \in K \tag{C1}$$

$$\sum_{k \in K} \sum_{r \in R_k} \pi_{jr}^k x_r^k \leq 1 \qquad \forall j \in J \tag{C2}$$

$$t_i - \sum_{k \in K} \sum_{r \in R_k} \tau_{ir}^k \pi_{ir}^k x_r^k - \sum_{i' \in I} a_{i'i} - \sum_{k \in K} d_i^k = 0 \qquad \forall i \in I \tag{C3}$$

$$d_i^k - D_i \sum_{r \in R_k} \pi_{ir}^k x_r^k \leq 0 \qquad \forall i \in I, k \in K \tag{C4}$$

$$\sum_{k \in K} d_{i'}^k - a_{i'i} + D_{i'} \sum_{k \in K} \sum_{r \in R_k} (\delta_{i'ir}^k x_r^k) \leq D_{i'} \qquad \forall i', i \in I \tag{C5a}$$

$$a_{i'i} - D_{i'} \sum_{k \in K} \sum_{r \in R_k} \delta_{i'ir}^k x_r^k \leq 0 \qquad \forall i', i \in I \tag{C5b}$$

$$a_{i'i} - \sum_{k \in K} d_{i'}^k \leq 0 \qquad \forall i', i \in I \tag{C5c}$$

and *Domain Constraints:*

$$-t_i + \alpha_i \sum_{k \in K} \sum_{r \in R_k} \pi_{ir}^k x_r^k \leq 0 \qquad \forall i \in I \tag{C6a}$$

$$t_i - \beta_i \sum_{k \in K} \sum_{r \in R_k} \pi_{ir}^k x_r^k \leq 0 \qquad \forall i \in I \tag{C6b}$$

$$y_i - y_{i'} \leq 0 \qquad \forall (i', i) \in P \tag{C7a}$$

$$t_{i'} - t_i + \lambda_{i'} + \tau_\infty (y_i - y_{i'}) + \epsilon_{i'i}^P y_{i'} \leq 0 \qquad \forall (i', i) \in P \tag{C7b}$$

$$y_{i'} - y_i = 0 \qquad \forall (i', i) \in S \tag{C8a}$$

$$t_{i'} - t_i + \epsilon_{i'i}^S y_{i'} = 0 \qquad \forall (i', i) \in S \tag{C8b}$$

Fig. 2. Mathematical model

a indicates a constraint type and b indicates an instance of that constraint), we use linear programming theory to derive the pricing subproblem for our model to be the problem of finding feasible routes r for agent k for which the following quantity is positive:

$$\begin{aligned} p_r^k = &-(u_k^1 + c_{1r}^k) + \sum_{i \in I} (D_i u_{ik}^4 - \alpha_i u_i^{6a} + \beta_i u_i^{6b}) \pi_{ir}^k \\ &+ \sum_{j \in J} (v_j - u_j^2) \pi_{i_o^j r}^k - \sum_{(i'i) \in P} (u_{i'i}^{8a} + \tau_\infty u_{i'i}^{8b}) \pi_{ir}^k \\ &+ \sum_{(i'i) \in P} (u_{i'i}^{8a} + (\tau_\infty - \epsilon_{i'i}^P - \lambda_{i'}^k) u_{i'i}^{8b}) \pi_{i'r}^k \\ &- \sum_{(i'i) \in S} (u_{i'i}^{9a} + \epsilon_{i'i}^S u_{i'i}^{9b}) \pi_{i'r}^k + \sum_{(i'i) \in S} u_{i'i}^{9a} \pi_{ir}^k \\ &+ \sum_{i \in I} \tau_{ir}^k u_i^3 \pi_{ir}^k + \sum_{i' \in I} \sum_{i \in I} (D_{i'} u_{i'i}^{5b} - D_{i'} u_{i'i}^{5a}) \delta_{i'ir}^k \end{aligned} \tag{2}$$

Solving this pricing subproblem involves searching for a route through a graph in which nodes represent subtasks and edges indicate that an agent can perform one subtask after another. Transition costs in the graph include node costs, edge costs, and costs that depend on the order of subtasks along partial routes leading up to a node. This last cost term arises due to the presence of cross-schedule dependencies, and significantly complicates the problem. A "profitable" feasible route is one that satisfies agent capacity constraints and branching constraints at the current branch-and-bound node, and whose overall cost is positive. Such routes, if they exist, could potentially increase the objective function of the solution. At each branch-and-bound node, we find such routes, add one or more of them to the master problem, re-solve the relaxed master problem, and continue the column generation process. When no such route is found, column generation ends at that node, and the branch-and-bound process continues.

To find feasible integer solutions, we adopt the following branching decisions, in the priority order listed below.

- *Branching on task pairs 'together'*: When there are a fractional routing variables such that two tasks occur together on one route but not on another, we branch by forcing the tasks to be on the same route ("together") in one branch and on different routes in the other branch.
- *Branching on subtask pair order*: When the fractional routing variables include two routes with the same subtasks performed in different orders, we constrain the subtasks to occur in a specific order in one branch and the opposite order in the other branch.
- *Branching on task agent*: When the fractional routing variables represent the same route performed by two different agents, we branch by forcing a task on that route to be performed by a given agent in one branch, and not by that agent in the other branch.

The optimal plan computed by the planner specifies an assignment of tasks to agents, and an order and schedule by which the agents should perform their assigned tasks. The schedule specifies precise task start times to ensure that cross-schedule constraints are satisfied. The computed optimal plan is transferred to the execution module.

C. Plan execution

Our plan execution strategy builds on the notion of plays, which are representations of deliberative multi-agent plans as coordinated sequences of team actions [2]. A play specifies a number of *roles*, and a role represents a sequence of actions to be executed by a single agent. Each agent on the team has a PlayExecutor, for executing actions, and a PlayManager, for monitoring current play participation and for handling all intra-play communications. For a given play, only one robot's PlayManager can have ownership of the play, with each of the other participating robots responsible for reporting their status to the play owner. Although initially formulated as a centralized, synchronous system in which the actions performed by each role are executed in lock step with other roles in the play, the most recent implementation allows for a more distributed approach in which plays are represented through a play specification strategy based on the Ruby scripting language [15]. This provides the flexibility for dynamic on-the-fly scripting of plays during execution. In our approach, the computed plan is automatically translated into a play whose roles comprise the individual single-agent plans computed by the planner. We further extend the plays paradigm such that it supports communication between roles to satisfy temporal constraints when required, but otherwise allows roles to be executed independently.

D. Plan translation for flexible execution

To make play execution flexible to operational variations, we need to be able to synchronize between different roles of the play when cross-schedule constraints need to be satisfied. We achieve this by one or more of the following synchronization-related actions:

send-message(key, msg): Sends a given message to a specified team member.

read-message(key): Checks for receipt of a specified message, waiting (up to a configured timeout) if that message has not yet been received.

check-message(key): Checks for receipt of a specified message, but does not wait if that message has not yet arrived.

read-message-by-time: Checks for receipt of a specified message, waiting up to a specified maximum end time, if that message has not yet been received.

wait-for-time: Waits until a specified time, if that time has not been reached, before beginning execution of the subtask.

Using the synchronization actions, the computed plans can be transformed to ensure satisfaction of cross-schedule temporal and time window constraints, as follows:

Precedence Constraints: For a precedence constraint such that task A must be performed before B, the agent that performs A sends a message, once that task is complete, to the agent assigned to B. Conversely, the agent assigned to B waits to receive a message concerning the successful completion of A before beginning execution of B. If the message indicates that A was successful, then the agent begins execution of B. Otherwise, it does not attempt to execute B but moves on to the next task in its schedule, removing from its schedule any additional tasks that depend on B and also notifying any other agents scheduled to execute tasks for which B is a pre-requisite. This enables graceful degradation of the plan in the event of task failure.

Synchronization Constraints: For a synchronization constraint such that tasks A and B must be performed at the same time, the agent assigned to each task sends a message, once it is ready to execute its task, to the agent assigned to the other task. Each agent then waits to receive the corresponding "Ready" message from the other agent, before beginning execution of its task. A message other than "Ready" indicates failure and the synchronized task is not executed. A configured timeout value indicates how long an agent will wait for a synchronization message.

Time Window Constraints: If there are time window constraints for a given subtask, the **wait-for-time** action is used

to ensure that a subtask is not executed before the beginning of its allowed time window. Similarly, a task will not be executed if an agent arrives at the location of the task after its time window. In the event that the task is the second task in a precedence constraint, or is involved in a synchronization constraint, a **read-message-by-time** action is used instead of the **read-message** action, to avoid waiting beyond the end of the allowed time window.

The synchronization actions must be used in a specific order to ensure feasible execution of the plan. Consider a segment of the computed plan that comprises traveling to a subtask location, optionally waiting for a specified amount of time, then performing the subtask:

```
travel-to <subtask location>
wait-till <subtask start time>
execute <subtask>
```

This plan segment is augmented with the synchronization actions as follows:

```
travel-to <subtask location>

for each precedence constr (A,B) where <subtask>=B
     read-or-wait-for-message("A-done")
for each synchronization constr (<subtask>,B)
     send-message ("<subtask>-ready", agent(B))
for each synchronization constr (<subtask>,B)
     read-or-wait-for-message("B-ready")

execute <subtask>

for each precedence constr (A,B) where <subtask>=A
     send-message ("<subtask>-done", agent(B))
```

If the subtask has an allowed time window, a **wait-till** action is inserted right after the travel-to action, and the **read-message-by-time** action is used instead of the **read-message** action. Note that for multi-way synchronization constraints (between more than two agents), the synchronization constraints between all pairs of agents in the group must be represented. Graceful degradation of the plan is enabled by skipping a subtask if the communicated message indicates that its required preceding or simultaneous subtasks cannot be executed. The agent then moves on to the next subtask in its plan. Furthermore, whenever a **read-message** or **read-message-by-time** action is needed prior to performing a task, the agent uses a **check-message** action before beginning travel to the task location, in order to avoid unnecessary travel if a message has been sent reporting unsuccessful completion of prerequisites of the task in question.

As the number of messages sent is proportional to the number of pairwise inter-task constraints in the problem, and the size of each message is only a few bytes, the bandwidth requirements of the approach is negligible. The approach is agnostic to the robot control architecture and, thus works well with heterogeneous agents. It can be further extended to handle imperfect communication between agents.

IV. Plan Generation Experiments and Results

While there exists a variety of problems with cross-schedule dependencies, for this paper we focus on a example scenario that involves providing transportation and sheltering assistance, in the event of an emergency, to clients with special needs. In this scenario, each client needs to be visited by a medical agent and then subsequently picked up and moved to an emergency shelter by a transportation agent. Consequently, there are temporal constraints between the medical visit and transportation of a client, and costs associated both with agents' travel times and with their waiting times. The term *delay penalty* describes the real-valued ratio between agent waiting costs and travel costs. For example, a delay penalty of 0 indicates that waiting time (e.g. to satisfy synchronization constraints) is not penalized, whereas a delay penalty of 0.5 indicates that it costs the agents half as much to wait as to travel for a given amount of time. Figure 3 illustrates an example problem with 6 clients, 2 shelters, 2 transportation agents and 1 medical agent.

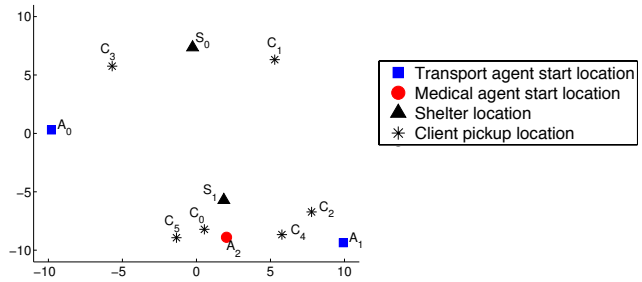
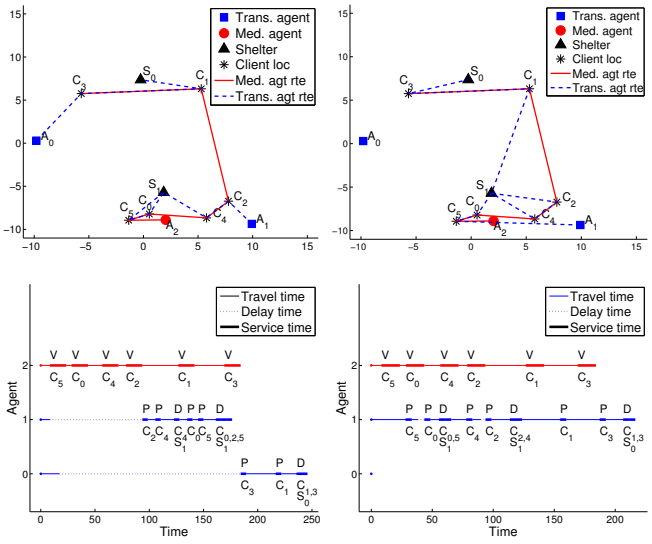


Fig. 3. Example problem with 6 clients, 2 transportation agents, 1 medical agent, and 2 shelters. Axes are in simulation distance units.

Figures 4(a) and 4(b) illustrate the optimal solution to the example problem with delay penalties of 0.0 and 0.5 respectively. In each case, the top subfigure illustrates the routes computed for each agent, while the bottom subfigure illustrates the computed schedules, coded by travel time, delay time, and service time. Service times are annotated with the subtask type (V for "visit", P for "pickup" and D for "dropoff") and client IDs. For drop-off subtasks, they are further annotated with the shelter ID. Table II summarizes the travel time, delay time and overall team cost for the two scenarios. With a delay penalty of 0, the algorithm computes the route that minimizes the agents' travel time. This however, results in a significant amount of delay for each transportation agent, which must wait for the medical agent to visit a client before it can transport that client. With a delay penalty of 0.5, the algorithm computes a new solution that has significantly less delay time. This new solution uses only one transportation agent, indicating that the algorithm effectively recognized that having only one medical agent creates a bottleneck such that extra transportation agents spend a lot of time waiting.

Figure 5 shows the value of the best solution and best bound over time for this problem, for delay penalties of 0.0 and 0.5. In both cases, the algorithm finds good solutions early, demonstrating anytime, bounded optimal behavior. However, a non-zero delay penalty has a significant impact on the time it takes to find and prove the optimal solution. This is because the algorithm must essentially evaluate the trade-off between travel time and delay time in potential



(a) Delay Penalty = 0.0 (b) Delay Penalty = 0.5

Fig. 4. Optimal routes (top) and schedules (bottom) for example problem with a delay penalty of (a) 0.0 and (b) 0.5. Distances and times are in simulation distance and time units respectively.

TABLE II

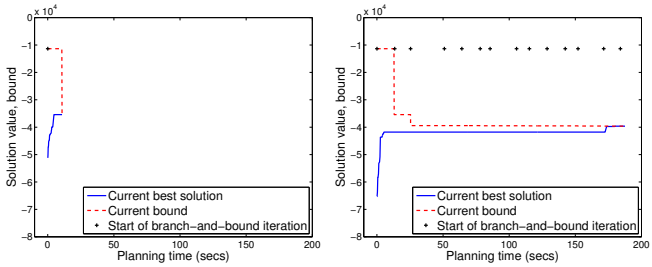
OPTIMAL SOLUTION TO EXAMPLE PROBLEM

Delay Penalty (dp)	**Total travel time** (t_t)	**Total delay time** (t_d)	**Total team cost** ($t_t + dp * t_d$)
0.0	204.14	252.32	204.14
0.5	241.71	9.06	246.24

(All times measured in simulation time units)

solutions it encounters during the solution process.

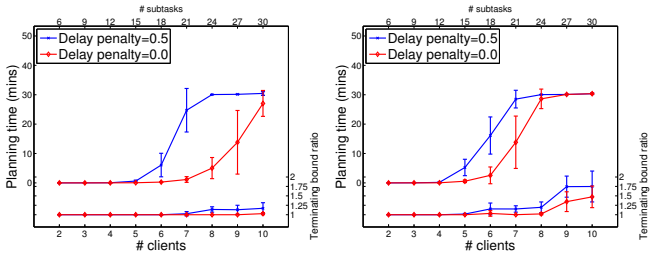
Figure 6 shows the total time to find and *prove* the optimal solution, averaged over 5 random instances of each problem configuration, and for problems with delay penalties of 0 and 0.5. Both for scenarios with precedence and those with synchronization constraints, the combinatorial nature of the problem is apparent in the rapid increase in the time needed to prove solution optimality as the problem size increases. Planning time was capped at 30 minutes, and the bottom graph indicates the ratio of the terminating solution to the terminating bound. A ratio of 1 indicates optimality.



(a) Delay penalty=0.0 (b) Delay penalty=0.5

Fig. 5. Example solution profiles

The figure also highlights the impact of delay penalties on problem difficulty. It illustrates that in the presence of precedence constraints (Figure 6(a)), problems that optimize a weighted sum of travel and delay time are significantly more difficult than those that optimize travel time alone. Problems with synchronization constraints (Figure 6(b)) are more difficult than ones with precedence constraints.



(a) Precedence constraints (b) Synchronization constraints

Fig. 6. Planning time to find and provide optimal solution

V. PLAN EXECUTION EXPERIMENTS AND RESULTS

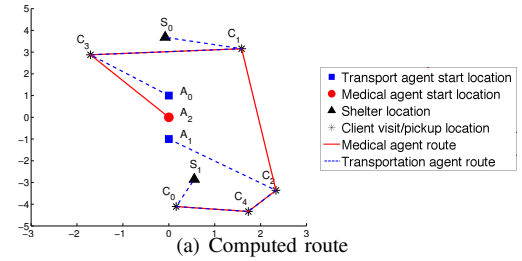
A. Experimental Setup

For our tests, we use a team of three (3) Pioneer P3-DX robots, one of which represents a medical agent while the other two (2) represent transportation agents. There are five (5) clients that require transportation assistance. In the first scenario, the medical visit is a two-part activity, the second part of which had to be scheduled at the same time as the pickup of the transportation service, as would occur if the agent performing the medical visit task also helps with client boarding. This is modeled as a synchronization constraint between the medical visit's second subtask and the pickup subtask. In the second scenario, the medical visit precedes the transportation service, resulting in precedence constraints.

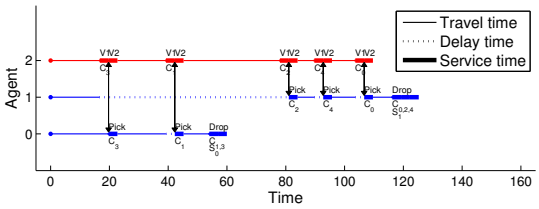
The experiments were run in a roughly 10m x 15m indoor space. Based on prior experimentation, robot capabilities and operational domain, we determined an average operational speed of the robots (0.2 m/s) to be used by the xTeam planner towards optimal plan generation. Furthermore, the expected execution times for each part of the medical visit tasks were scaled down to be comparable to the travel times in the indoor test area. Consequently, the expected execution times for each part of the medical visit tasks was 3s and for a total medical visit time of 6s per client. The pickup and drop-off tasks were each specified to require 3s each.

The routes computed by the planner were the same for both the synchronization and the the precedence scenarios (see Figure 7(a)). The computed timeline for the synchronization scenario, showing travel time, waiting time, and task execution time for each agent, is illustrated in Figure 7(b), while that for the precedence scenario is in Figure 7(c).

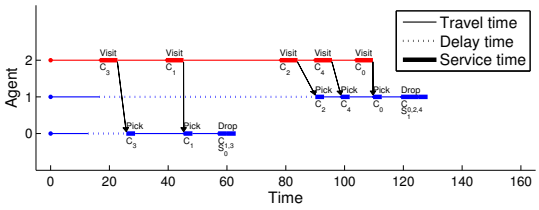
To validate that constraints are not violated despite deviations from the plan conditions during execution, the robots were tested for three different execution cases using the same generated optimal plan. In the first case, the durations of both types of tasks were as expected. In the second, the first part of



(a) Computed route



(b) Timeline for synchronization scenario



(c) Timeline for precedence scenario

Fig. 7. Optimal plan computed for the experiment problem

each medical visit task was shorter than expected (1s instead of 3s), while in the third, the first part of each medical visit task was longer than expected (5s instead of 3s). The agents executed their plans in one of 3 modes:

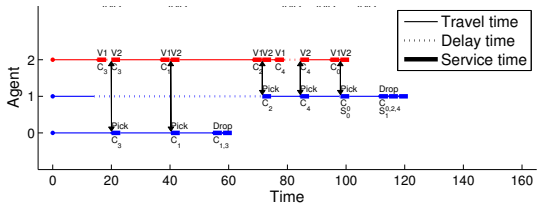
Flex mode: This mode represents the flexible execution strategy described in this paper, in which the agents relax the precise schedules computed by the planner and exchange synchronization messages as needed to determine when subtasks can be feasibly executed.

Fixed-start mode: In this mode, the agents do not exchange synchronization messages during plan execution. Each agent instead attempts to adhere to the subtask start times specified by the plan. If an agent reaches a location before the specified subtask start time, it waits before beginning execution. If it arrives late, it immediately executes the subtask and then moves on to the next item in its plan.

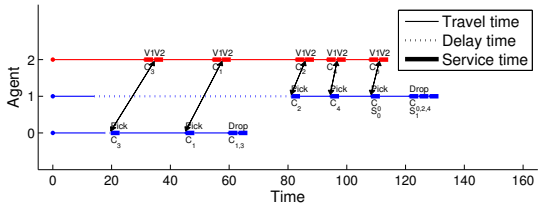
Fixed-wait mode: The agents do not exchange synchronization messages in this mode either. Instead, they adhere strictly to the wait times specified by the plan. Whenever an agent arrives at a location, it waits for precisely the amount of waiting time, if any, specified by the plan. It then executes the subtask and moves on to the next item in the plan.

For each of the two problem scenarios (synchronization and precedence) and each of the three execution cases (normal-length visits, shorter-length visits, and longer-length visits), the team of robots conducted three runs in each of the three possible execution modes (flex mode, fixed-start mode, and fixed-wait mode), for a total of 54 experimental runs. During execution, the agents' travel speeds varied slightly, due to "real world" mobility considerations such as an unexpected obstacle, inter-robot path interference, noisy sensors etc. To evaluate the performance of our execution strategy, we analyze the system under two operational modes, that of constraint satisfaction and graceful degradation.

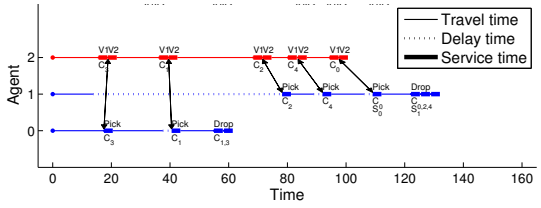
B. Constraint satisfaction results



(a) Timeline for sample "flex" execution mode



(b) Timeline for sample "fixed-start" execution mode



(c) Timeline for sample "fixed-wait" execution mode

Fig. 8. Sample execution timelines for the synchronization scenario

Figure 8 shows the timelines for sample runs of each of the three execution strategies for the case with synchronization constraints and normal-length visits. In the sample run using the flex mode, the relevant subtasks were perfectly synchronized. To achieve this, a short wait time was automatically inserted between the two parts of the medical visit to client C_3, the first client visited by the medical agent (Agent 2). Similarly, waiting time was inserted between the two parts of the visit to client C_4, because the travel time of the transportation agent, Agent 1, with which the medical agent had to synchronize, was longer than expected. For the sample runs using the "fixed-start" and "fixed-wait" execution modes, most of the subtasks to be synchronized were considerably mis-aligned, due to execution time variations in travel speed.

For corresponding runs with precedence constraints, we found that the constraints were often satisfied using all three execution strategies. This is because precedence constraints allow more flexibility than synchronization constraints, and because the medical agent's travel time was generally not much worse than expected. However, the plan was completed earlier in the "flex" execution mode than it was in the other two modes and hence was more efficient.

In the event of a constraint violation, we computed an associated "constraint violation time", given by:

Synchronization Constraints: If subtasks A and B are supposed to be executed together, than the constraint violation time is the absolute value of the difference between the start times of subtask A and subtask B.

Precedence Constraints: If subtask A is supposed to be done before B, but B is started before A, then the constraint violation time is the amount of time between the start time of B and the completion time of A. If B is started after the completion time of A, the constraint violation time is 0.

The constraint violation time for an entire plan execution is the sum of the violation times for each constraint. Our test runs each have 5 constraints. Figure 9 shows the constraint violation time per run, averaged over 3 runs for each execution mode and scenario. It illustrates that the "flex" execution mode effectively prevents constraint violations.

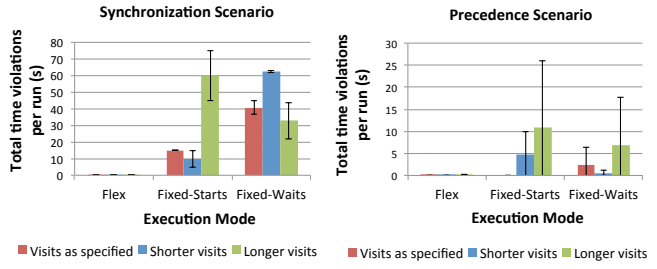


Fig. 9. Constraint violations for synchronization scenario (left) and for precedence scenario (right)

C. Graceful degradation results

In certain situations, graceful degradation allows the system to isolate failure states during execution and to continue executing the rest of the plan. We tested this feature of our approach by simulating a failure or task cancellation for the second client visited by the medical agent. Once the medical agent detects failure, it communicates to cancel the associated transportation task for the client. Consequently, one of two situations arises. In the first situation, termed the "abort" scenario, the task cancellation is communicated to the transportation agent only after the medical agent reaches the intended client location. Thus, each agent still does as much traveling as before and cannot reduce the overall length of the remaining plan. Alternately, in the second situation, termed the "skip" scenario, the medical agent learns of the task cancellation before it sets off to the client location, and communicates this information to the transportation agent. Consequently, both the medical and transportation agents skip visiting the client location altogether. Thus, the overall travel time and plan completion time for the team is reduced. Table III summarizes the medical agent travel times, transport agent travel and waiting times, and plan completion times for experimental runs of these two scenarios.

TABLE III
GRACEFUL DEGRADATION

	Abort Scenario	Skip Scenario
Medical agent travel time (s)	77.15 $\pm$ 8.08	53.67 $\pm$ 1.23
Transport agent travel time (s)	98.69 $\pm$ 7.07	75.10 $\pm$ 7.94
Transport agent waiting time (s)	63.19 $\pm$ 18.96	37.52 $\pm$ 3.78
Plan completion time (s)	120.81 $\pm$ 9.58	93.93 $\pm$ 0.82

VI. Conclusion and Future Work

We present an approach for generating optimal plans and for flexible execution of multi-robot plans with cross-schedule temporal constraints. The approach includes an anytime, bounded optimal planner and a flexible execution strategy that ensures that temporal ordering constraints are satisfied, even in the face of real-world variations during plan execution. The approach also allows for graceful degradation of the plan when tasks fail or cannot be executed. Our future work will extend the notion of graceful degradation to re-planning when tasks fail or new tasks come in, thus closing the loop between the planning and execution modules.

References

[1] G. A. Korsah, "Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies," Ph.D. dissertation, Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, Jan 2011.

[2] M. Bowling, B. Browning, and M. Veloso, "Plays as effective multiagent plans enabling opponent-adaptive play selection," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004.

[3] M. B. Dias, "Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments," Ph.D. dissertation, Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, Jan 2004.

[4] B. P. Gerkey and M. J. Matarić, "Sold!: auction methods for multirobot coordination," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, pp. 758–768, Oct 2002.

[5] A. Whitten, H.-L. Choi, L. Johnson, and J. How, "Decentralized task allocation with coupled constraints in complex missions," in *American Control Conference (ACC), 2011*, July 2011, pp. 1642 –1649.

[6] H.-L. Choi, L. Brunet, and J. How, "Consensus-based decentralized auctions for robust task allocation," *Robotics, IEEE Transactions on*, vol. 25, no. 4, pp. 912 –926, aug. 2009.

[7] M. Koes, I. Nourbakhsh, and K. Sycara, "Constraint optimization coordination architecture for search and rescue robotics," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2006*, May 2006, pp. 3977–3982.

[8] D. Bredström and M. Rönnqvist, "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints," *European Journal of Operations Research*, vol. 191, pp. 19–31, 2008.

[9] R. Alami and S. S. d. C. Bothelho, "Plan-based multi-robot cooperation," in *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents,*, 2002, pp. 1–20.

[10] S. Joyeux, R. Alami, S. Lacroix, and R. Philippsen, "A plan manager for multi-robot systems," *International Journal of Robotics Research*, vol. 28, pp. 220–240, February 2009.

[11] P. Pirjanian, T. Huntsberger, and A. Barrett, "Representation and execution of plan sequences for multi-agent systems," in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[12] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," in *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, 1989, pp. 83–93.

[13] S. Smith, A. T. Gallagher, T. L. Zimmerman, L. Barbulescu, and Z. Rubinstein, "Distributed management of flexible times schedules," in *2007 Intl conf on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2007.

[14] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Op. Res.*, vol. 46, pp. 316–329, 1998.

[15] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz, "Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks," in *International Conference on Robotics and Automation*, May 2006, pp. 570 – 575.