xBots: An Approach to Generating and Executing Optimal Multi-Robot Plans with Constraints

G. Ayorkor Korsah, Balajee Kannan, Brett Browning,
M. Bernardine Dias *

CMU-RI-TR-11-25

August 2011

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

*Balajee Kannan, Brett Browning, and M. Bernardine Dias are affiliated with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {bkannan,brettb,mbdias}@ri.cmu.edu. G. Ayorkor Korsah is an Assistant Professor at Ashesi University College, Accra, Ghana ayorkor@alumni.cmu.edu.

Abstract

In this report, we present an approach to optimal planning and flexible execution for a set of spatially distributed tasks related by temporal ordering constraints such as precedence, synchronization, or non-overlapping constraints. We integrate an optimal planner for task allocation and scheduling with cross-schedule dependencies with a flexible, distributed plan execution strategy. The integrated system performs optimal task allocation and scheduling for tasks related by temporal constraints, and ensures that plans are executed smoothly in the face of real-world variations in operation speed and task execution time. It also ensures that plan execution degrades gracefully 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 focus on the flexible execution strategy and illustrate that it effectively enables execution of the optimal plan and prevents constraint violations. The overall approach is thus demonstrated to be effective for constrained planning and execution in the face of real-world variations.

Contents

1 Introduction

Multi-robot teams will increasingly be used to address heterogeneous spatially distributed tasks in domains where no single team member has the capabilities or the reliability to effectively execute all tasks. As the complexity of the domain increases, coordination becomes challenging given the mixture of spatial, temporal and team capability reasoning required. In these scenarios, robust coordination techniques are essential for ensuring that tasks are successfully executed. The nature of the coordination problem to be solved is highly domain dependent. In some cases, tasks are distributed among robots for independent execution, whereas in others the robots require constant and tightly-coupled interaction amongst themselves to successfully complete the tasks. The work presented in this report enables task execution in domains that have spatially distributed high-level tasks related by temporal constraints. Such problems fall in the category of those with *cross-schedule dependencies* [1] due to the interdependence between the schedules of different agents. This is a rich and diverse category of problems that spans domains such as emergency assistance, agriculture, construction, and planetary exploration. Consider an example scenario involving emergency assistance where individuals with special needs must be sheltered in the event of an emergency. Each client with special needs might need to be visited by a medical agent and then moved to an emergency shelter by a transportation agent. Consequently, there are temporal constraints (precedence or synchronization) that must be satisfied between the medical visit and subsequent transportation. Thus, the problem requires joint coordination of transportation and medical agents, subject to cross-schedule constraints.

With a focus on plan quality, the goal is to optimally plan for task allocation and scheduling for the heterogeneous team, and to subsequently execute the computed plan, while ensuring the temporal constraints between the tasks are satisfied. These constraints must be satisfied even in the face of execution-time variations in the operating domain.

The main contribution of this report is xBots – an integrated approach to optimal planning and flexible execution for a collection of multi-robot tasks with cross-schedule constraints. The system combines the optimal xTeam planner [1] for task allocation and scheduling with a new flexible execution approach based on the plays paradigm [2]. We experimentally demonstrate the effectiveness of xBots in computing and executing optimal plans without constraint violations on a team of indoor robots.

The organization of the remainder of the report is as follows. Section 2 presents related work while Section 3 details of the xBots approach. Section 4 outlines the experiment setup and present our results and analysis in section 5. Finally, we end with a summary and discussion of future work in section 6.

2 Related Work

There is a diverse literature on multi-robot coordination, planning and execution. We will briefly review some key planning and execution frameworks that address the issue of multi-robot cooperation during execution. A review of the literature, however, reveals a dearth of discussion on multi-robot execution of constrained optimal plans.

Several approaches use negotiation to formulate team plans and ensure conflict-free execution. For example, Alami et al. [3, 4, 5, 6] present a framework for multi-robot cooperation comprising the M+NTA scheme for Negotiation for Task Achievement and the M+CTA scheme for Cooperative Task Achievement. This framework focuses on cooperation to achieve independent goals. The independent goals are first allocated to robots using the market-based M+NTA scheme. The M+CTA scheme provides a means for robots to detect and treat resource conflicts by inserting temporal order constraints between two actions belonging to two robots. As another example, Joyeux et al [7] present a shared "plan database" for building, negotiating, and executing plans in a multi-robot context.

Other relevant work on multi-robot plan execution includes CAMPOUT by Pirjanian et al [8], 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. Additionally, Osherenko [9] proposes a plan representation for multi-agent systems accomplishing cooperative tasks in the real world. Their approach combines the ACT formalism [10] for representing hybrid (deliberative and reactive) multi-agent plans with ICL [11], an inter-agent communication language.

None of these approaches address optimal planning and flexible execution of high-level tasks related by temporal constraints. Furthermore, none of the existing strategies can easily translate generated plans, optimal or otherwise, for flexible execution.

Simple Temporal Networks (STNs) [12] are often used to represent flexible time plans. These networks, instead of representing the start time of tasks as fixed time points, represent them as a time window, or a range of feasible times. A decision on the exact start time of a task is not made during the planning stage, but is delayed until execution time. As time progresses and tasks are completed, the STN is updated and any changes to allowed time windows are propagated through the network. This has the advantage that at any point in time, the consistency of the remainder of the plan can be verified. Simple Temporal Networks have been used to enable flexible scheduling of the plans of single agents [13], as well as individual agents operating as part of a team [14]. While STNs are a fair representative of flexible plans, they do not by themselves represent optimal plans. The use of STNs is complementary to our approach which focuses on ensuring satisfaction of cross-schedule dependencies during execution of a pre-computed optimal plan, with minimum communication between agents.

In summary, currently there exists no strategy for integrating optimal planning with flexible execution strategies to ensure cross-schedule constraints satisfaction for multi-robot teams.

3 Approach

In this section, we outline xBots, our approach for optimal planning and flexible plan execution for multi-robot teams. The overall approach comprises a planning module and an execution module, illustrated in Figure 1.

Given prior information, such as estimates of agent speeds and task execution times, the planning module computes an optimal allocation of tasks to agents and a schedule according to which the tasks must be executed. The current implementation of the

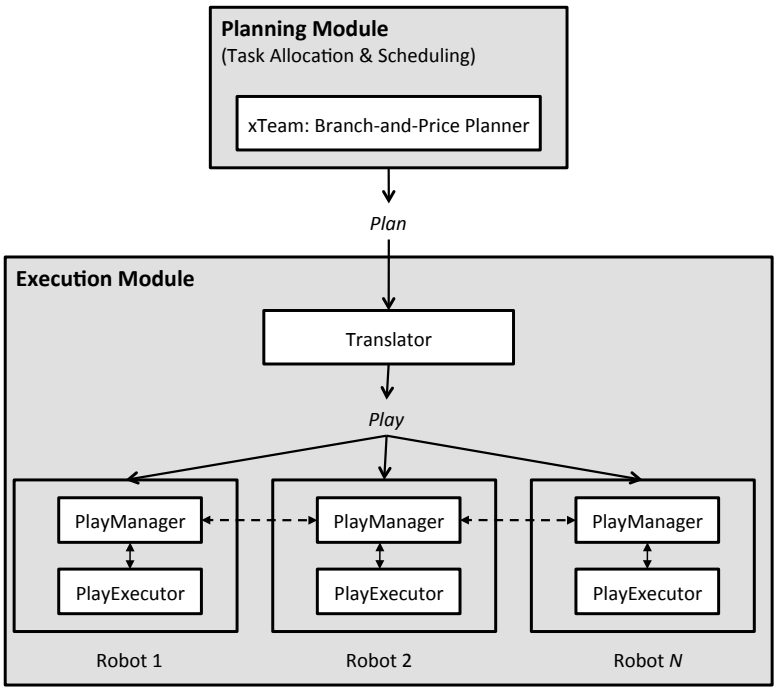


Figure 1: xBots approach for optimal planning and flexible execution

xBots approach uses a specific optimal planner, xTeam [1] to generate the constrained team plans for execution. The computed schedule for each agent includes task start times, and corresponding waiting times needed to ensure that inter-task constraints as well as time-window constraints are satisfied. This plan is then transferred to the execution module.

In an ideal world where there is no variation in parametric values, 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. It is thus necessary for the agents to have an awareness of the relevant high-level constraints impinging on their part of the team plan and a strategy to enable flexible and feasible execution of the computed team 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 in which agents are loosely-coupled and intermittently synchronize with each other. This strategy is an extension of the *plays* [2] paradigm. We also outline an automated approach for converting the generated optimal plans for a multi-robot team to a flexible plan that can accommodate real-world execution-time variations. This is achieved by a "Translator" sub-module that provides an interface for interaction between the planning and execution, thereby allowing the execution strategy to be decoupled from the high-

level planner and optionally used to with other solvers.

The remainder of this section briefly describes the xTeam planner used for plan generation, the play-based plan execution approach, and the process of automated plan translation for flexible execution.

3.1 Plan generation

Our implementation uses xTeam [1], a centralized mathematical programming based planning approach that enables the computation of optimal plans for the task allocation and scheduling problem with cross-schedule dependencies. xTeam is a custom *branch-and-price* [15] algorithm that computes progressively better solutions, with bounds on quality, until it returns a provably optimal solution. This 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 decoupling of the planning and the execution modules in the xBots approach makes it possible to accommodate other planners in the future. The choice of the xTeam planner for the initial implementation is due to the capability of optimal planning for problems with cross-schedule constraints.

3.2 Plan execution

As mentioned above, our plan execution strategy builds on the notion of plays, originally developed for the robot soccer domain. A play represents a deliberative multi-agent plan as a coordinated sequence 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 robots participating in the play 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 [16]. This provides the flexibility for dynamic on-the-fly scripting of plays during execution. In this work, We further extend the plays paradigm to support communication between roles to satisfy synchronization and precedence constraints when required while allowing each agent to otherwise execute its role independently.

3.3 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. For this purpose, each agent has a messaging daemon that receives and stores messages on its behalf until the messages are needed.

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

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 already 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 task B, the agent that performs task A sends a message, once that task is complete, to the agent assigned to task B. Conversely, the agent assigned to task B waits to receive a message concerning the successful completion of task A before beginning execution of task B. If the message indicates that task A was successful, then the agent begins execution of task B. Otherwise, if task A was not executed successfully, it does not attempt to execute task 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. Similar to the precedence scenario, 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.

Non-overlapping Constraints: For a non-overlapping constraint, the executions of tasks A and B must not overlap, although it does not matter which is done first. In the plan computed by the xTeam planner, however, a commitment has been made as to which of the two tasks will be performed first. As such, at execution time, non-overlapping constraints can be treated like precedence constraints.

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 prior to performing a task, the agent uses a **check-message** action before traveling to the task location, in order to avoid unnecessary travel if a message has been sent reporting unsuccessful completion of the task in question.

As the number of messages sent is proportional to the number of pairwise intertask constraints in the problem, and the size of each message is only a few bytes, the bandwidth requirements of the approach is negligible. Furthermore, if there are no time window constraints on a subtask, no initial clock synchronization is required between the agents, thereby further reducing the communication overhead. Additionally, the approach is agnostic to the robot control architecture and, as such works well with heterogeneous agents. It can be further extended to handle imperfect communication between pairwise agents whose schedules are related by temporal constraints.

In our domain, each agent largely executes its role independently, and we use intermittent communication between roles, as described, to enable synchronized and coordinated behavior when required for specific subtasks. The computed plan is automatically translated into a play whose roles comprise the individual single-agent plans computed by the planner, augmented with the synchronization constructs previously described.

4 Experimental setup

To demonstrate our approach, we use the previously outlined scenario of providing transportation assistance to clients with special needs in the event of an emergency. For our tests, we use a team of three (3) Pioneer P3-DX robots (Figure 2), 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, modeling a situation where the agent performing the medical visit task also helps load the client into the transportation vehicle. This is modeled as a synchronization constraint between the second subtask of the medical visit subtask and the pickup subtask. In the second scenario, the medical visit precedes the transportation service, resulting in precedence constraints.

Figure 2: Pioneer robots

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, to reduce wait time and increase operational efficiency, the expected execution times for each part of the medical visit tasks were scaled down to be comparable to the travel times. Consequently, the expected execution times for each part of the medical visit tasks was 3 seconds and for a total medical visit time of 6 seconds per client. The pickup and drop-off tasks were each specified to require 3 seconds each.

The routes computed by the planner were the same for both the synchronization and the the precedence scenarios (see Figure 3(a)). The medical visit agent, A_2 follows the route from its start location through the sequence of client locations, C_3, C_1, C_2, C_4, C_0. The transportation agent A_0 picks up clients C_3 followed by C_1, and drops them both off at the shelter S_0. The second transportation agent, A_1, picks up the clients C_2, C_4, and C_0, and drops them off at the shelter S_1. Although both scenarios have the same computed routes, they have different timelines. The computed timeline for the syn-

chronization scenario, showing travel time, waiting time, and task service/execution time for each agent, is illustrated in Figure 3(b), while that for the precedence scenario is shown in Figure 3(c).

In prior work [1], we have performed detailed analysis on the optimality of the generated plans and hence focus our testing on execution performance. To validate that the 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 for all three cases. In the first, the durations of both types of tasks were as expected. In the second, the first part of each medical visit task was shorter than expected (resulting in a total visit time per client of 4s instead of 6s), while in the third, the first part of each medical visit task was longer than expected (resulting in a total visit time per client of 8s). The agents executed their plans in one of 3 modes:
Flex mode: This mode represents the flexible execution strategy described in this report, 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 the second execution strategy, 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 start time of the task, it waits until the specified start time before beginning execution of the subtask. If it arrives at a location after the specified start time of the subtask, it immediately executes the subtask and then moves on to the next item in its plan.
Fixed-wait mode: In the third execution strategy as well, the agents do not exchange synchronization messages during during plan execution. Instead, the agents adhere strictly to the subtask 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.

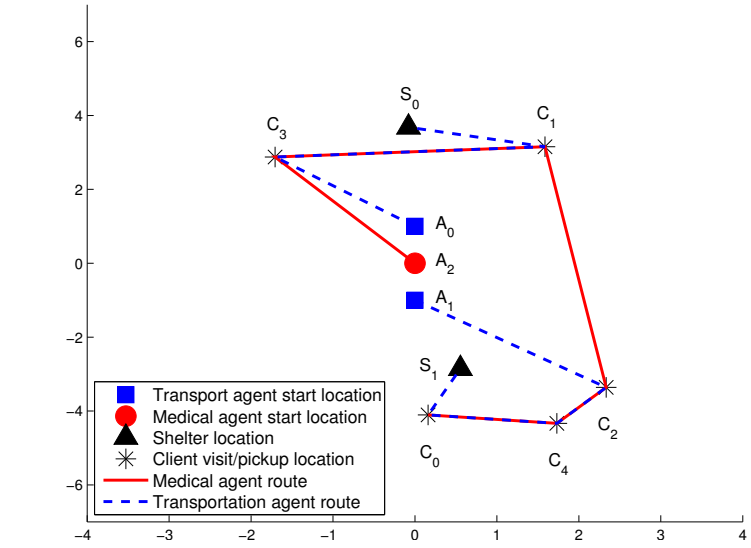
For each of the two (2) problem scenarios (synchronization and precedence) and each of the three (3) execution cases (normal-length visits, shorter-length visits, and longer-length visits), the team of robots conducted three (3) runs in each of the three (3) possible execution modes (flex mode, fixed-start 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 obstacle in the environment, path interference between the robots, noisy sensors etc.

5 Results and analysis

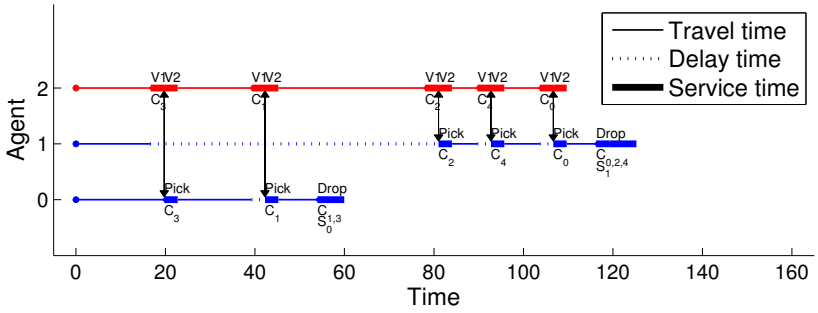
To evaluate the performance on our execution strategy, we analyze the system under two operational modes, that of constraint satisfaction and graceful degradation.

5.1 Constraint satisfaction

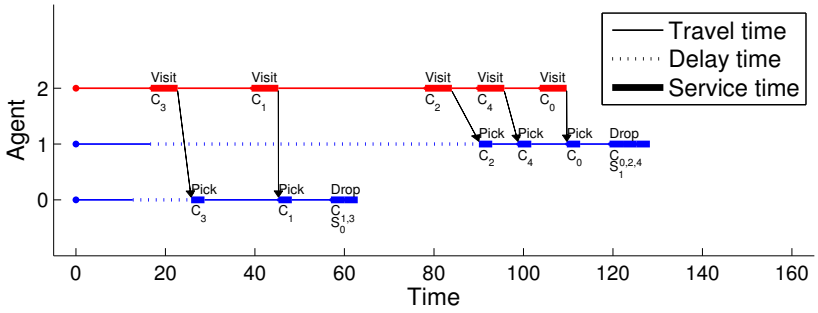
Figure 4 shows the timelines for sample runs of each of the three execution strategies for the case with synchronization constraints and normal-length visits. In the sample



(a) Computed route



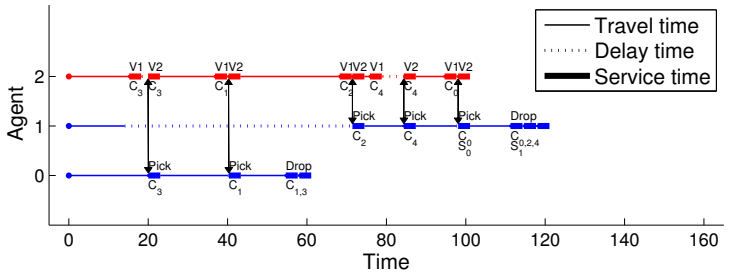
(b) Timeline for synchronization scenario



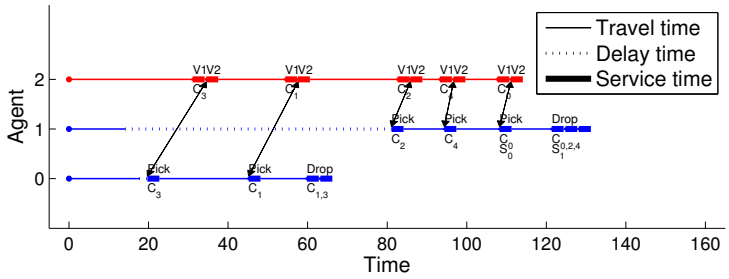
(c) Timeline for precedence scenario

Figure 3: Optimal plan computed for the experiment problem

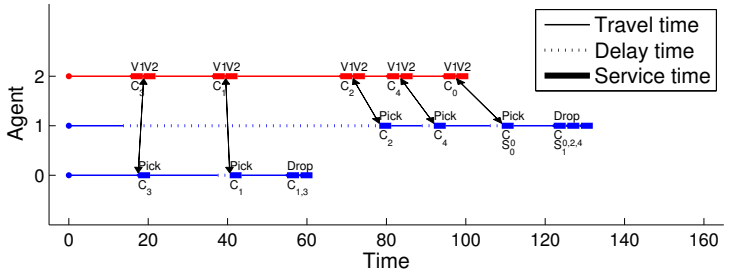
run using the flex mode, the relevant subtasks were perfectly synchronized. To achieve this, it can be seen that a short wait time was inserted between the two parts of the medical visit to client C_3, which is the first client visited by the medical agent (Agent 2). Similarly, waiting time was inserted 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 respectively, most of the subtasks to be synchronized were considerably mis-aligned, due to execution time variations in travel speed.



(a) Timeline for sample "flex" execution mode



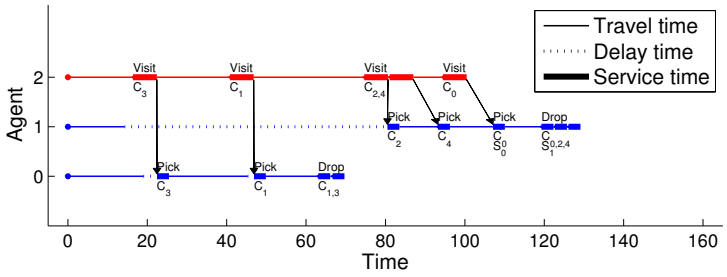
(b) Timeline for sample "fixed-start" execution mode



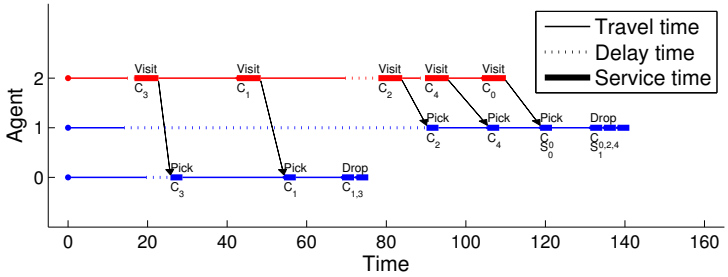
(c) Timeline for sample "fixed-wait" execution mode

Figure 4: Sample execution timelines for the synchronization scenario

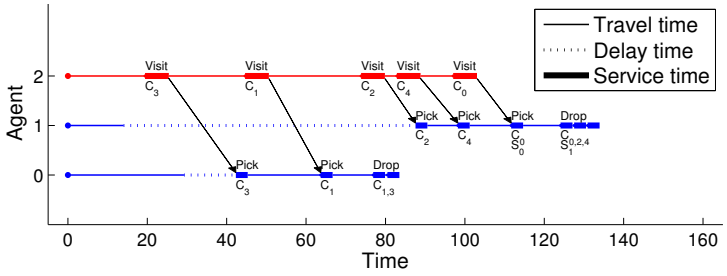
Figure 5 shows the timelines for sample runs of each of the three execution strategies for the case with precedence constraints and normal-length visits. For these sample



(a) Timeline for sample "flex" execution mode



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



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

Figure 5: Sample execution timelines for the precedence scenario

runs, all the precedence constraints were satisfied using all three execution strategies. This was because the medical agent's travel time was as expected or better than expected in all three runs, so the medical agent was always able to complete its task before the transportation agent performed the corresponding pickup task. However, the plan was completed earlier in the "flex" execution mode than it was in the other two modes and therefore 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.

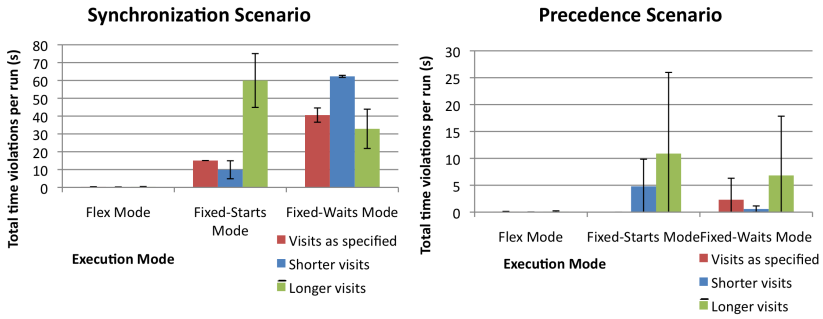


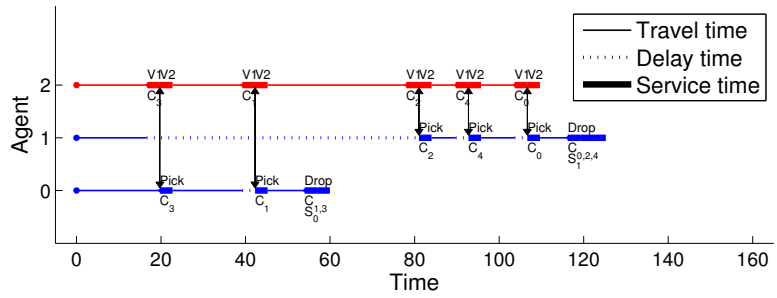
Figure 6: Constraint violations for synchronization scenario (left) and for precedence scenario (right)

Precedence Constraints: If subtask A is supposed to be done before subtask B, but subtask B is actually 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.

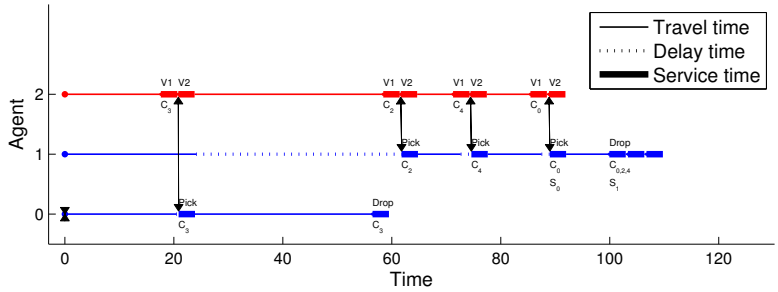
The constraint violation time for an entire plan execution is the sum of the constraint violation times for each constraint. In our test cases, there are 5 constraints for each run. Figure 6 shows the average constraint violation time per run, averaged over 3 runs for each execution mode and scenario. The figures illustrate that the "flex" execution mode effectively prevents constraint violations and as such is a simple and effective approach to flexible execution of optimal plans with cross-schedule constraints.

5.2 Graceful degradation

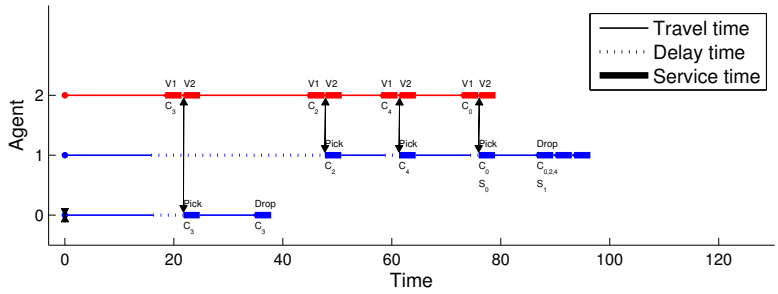
In certain situations, graceful degradation allows the system the flexibility to isolate failure states during execution and to continue executing the rest of the plan, allowing for an improved if sub-optimal execution performance. Figure 7 demonstrates the utility of the graceful degradation property of the execution framework for a sample run. The generated plan (see Figure 7(a)) dictates client C_1 is the second client visited by the medical agent. However, we simulate an execution failure for the medical agent. Once the medical agent acknowledges failure, it terminates the associated transportation task for the client. Consequently, one of two situations arises. In the first situation, depicted by figure 7(b), 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 rather than attempting to reduce the overall operational distance of the remaining plan. Alternately, in the second situation, figure 7(c), the medical agent acknowledges failure before it sets off to the client location and communicates to the transportation agent. Consequently, both the medical and transportation agents skip visiting the client location altogether. Thus, the overall travel distance for the team is reduced.



(a) Planned timeline



(b) Timeline when the medical task for C1 fails/is aborted



(c) Timeline when the medical task for C1 is skipped

Figure 7: Graceful degradation - when a task fails or is skipped, its dependent tasks are removed from the plan, and the agents move on to perform the remaining tasks.

6 Conclusion and Future Work

We present an approach for generating optimal plans and flexible execution of multi-robot plans with cross-schedule temporal constraints. The flexibility of the strategy 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 address the question of re-planning when tasks fail or

new tasks come in, thus closing the loop between the planning and execution modules. From a planning perspective, we will relax current assumption regarding task execution ordering for non-overlapping constraints. This will provide the execution strategy with an additional layer of flexibility with regards to the task execution ordering.

7 Acknowledgments

The authors would like to acknowledge members of the rCommerce Lab at Carnegie Mellon University for their important contributions towards hardware and software development. This work was sponsored in part by the Qatar National Research Fund under contract NPRP 1-7-7-5, and by the Boeing company under contract CMU-BA-GTA-1. The content of this publication does not necessarily reflect the position or policy of the sponsors and no official endorsement should be inferred.

References

[1] G. A. Korsah, "Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 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] 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,*. London, UK: Springer-Verlag, 2002, pp. 1–20. [Online]. Available: http://portal.acm.org/citation.cfm?id=647355.724921

[4] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and R. F., "Multi-robot cooperation in the martha project," *Robotics & Automation Magazine*, vol. 5, no. 1, Mar 1998), pages = 36–47.

[5] R. Alami, F. Ingrand, and S. Qutub, "A scheme for coordinating multi-robot planning activities and plans execution," in *In Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98*, 1998.

[6] F. Gravot and R. Alami, "An extension of the plan-merging paradigm for multi-robot coordination." in *ICRA'01*, 2001, pp. 2929–2934.

[7] 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. [Online]. Available: http://portal.acm.org/citation.cfm?id=1497541.1497547

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

[9] A. Osherenko, "Plan representation and plan execution in multi-agent systems for robot control," in *In workshop of KI 2001 Joint German/Austrian Conference on Artificial Intelligence*, 2001.

[10] D. E. Wilkins and K. L. Myers, "A common knowledge representation for plan generation and reactive execution," *Journal of Logic and Computation*.

[11] D. L. Martin, A. J. Cheyer, and D. B. Moran, "The open agent architecture: A framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 91–128, 1999.

[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*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 83–93. [Online]. Available: http://portal.acm.org/citation.cfm?id=112922.112931

[13] P. H. Morris, N. Muscettola, and T. Vidal, "Dynamic control of plans with temporal uncertainty," in *IJCAI*, 2001, pp. 494–502.

[14] 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.

[15] 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," *Operations Research*, vol. 46, pp. 316–329, 1998.

[16] 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.