Coordinated Hospital Patient Scheduling

Keith Decker and Jinjiang Li
Department of Computer and Information Sciences
University of Delaware
(decker,jli)@cis.udel.edu

Abstract

Hospital Patient Scheduling is an inherently distributed problem because of the way real hospitals are organized. As medical procedures have become more complex, and their associated tests and treatments have become interrelated, the current ad hoc patient scheduling solutions have been observed to break down [15]. We propose a multi-agent solution using the Generalized Partial Global Planning (GPGP) approach that preserves the existing human organization and authority structures, while providing better system-level performance (increased hospital unit throughput and decreased patient stay time). To do this, we extend GPGP with a new coordination mechanism to handle mutually exclusive resource relationships. Like the other GPGP mechanisms, the new mechanism can be applied to any problem with the appropriate resource relationship. We evaluate the this new mechanism in the context of the hospital patient scheduling problem, and examine the effect of increasing interrelations between tasks performed by different hospital units.

1. Introduction

Generalized Partial Global Planning (GPGP) is a task environment centered approach to coordination [8, 5]. The basic idea is that each agent constructs its own local view of the structure and relationships of its intended tasks. This view may then be augmented by information from other agents, and it may change in other ways dynamically over time. The GPGP approach uses a set of individual *coordination mechanisms* to help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. These commitments result in more coherent, coordinated behavior [3, 2, 11]. No one coordination algorithm will be appropriate for all task environments, but by selecting from a set of possible coordination mechanisms we can create a wide set of different coordination responses.

We have demonstrated the usefulness of the GPGP approach using five coordination mechanisms drawn from Durfee's original PGP work [9] in several domains—distributed vehicle monitoring [6], distributed data processing [16], and in randomly generated problems [8]. This paper contains a study of GPGP as applied to the hospital patient scheduling domain. The domain is quite interesting because it is inherently distributed due to the existing human authority structure of a hospital.

Although we can apply the five existing GPGP coordination mechanisms to the hospital patient scheduling problem directly, the hospital patient scheduling problem contains a task relationship that is not handled by any of the existing coordination mechanisms, namely, that the hospital patient is a mutually exclusive resource. To address this new relationship, first we must define it formally as a resource constraint in our task structure representation language (TÆMS [7, 18]) and then we define a new GPGP coordination mechanism to handle the new task relationship. We show how the existing practices can be captured using GPGP and *removing* some of the mechanisms. GPGP *with* our new resource-constraint mechanism outperforms existing practice on modern, complex procedures in two ways: increased patient throughput, and decreased patient stay time. Historically, hospital procedures were not as mutually interrelated as they are becoming today. We also demonstrate that the existing practice produces similar results to our new mechanism when few inter-unit task relationships exist.

The paper is arranged as follows. First we discuss the hospital patient scheduling problem. Next we briefly discuss GPGP, and focus on the new resource constraint coordination mechanism. Finally, we present our experimental results.

2. The Hospital Scheduling Problem

Our model is drawn from a case study of an actual hospital [15]:

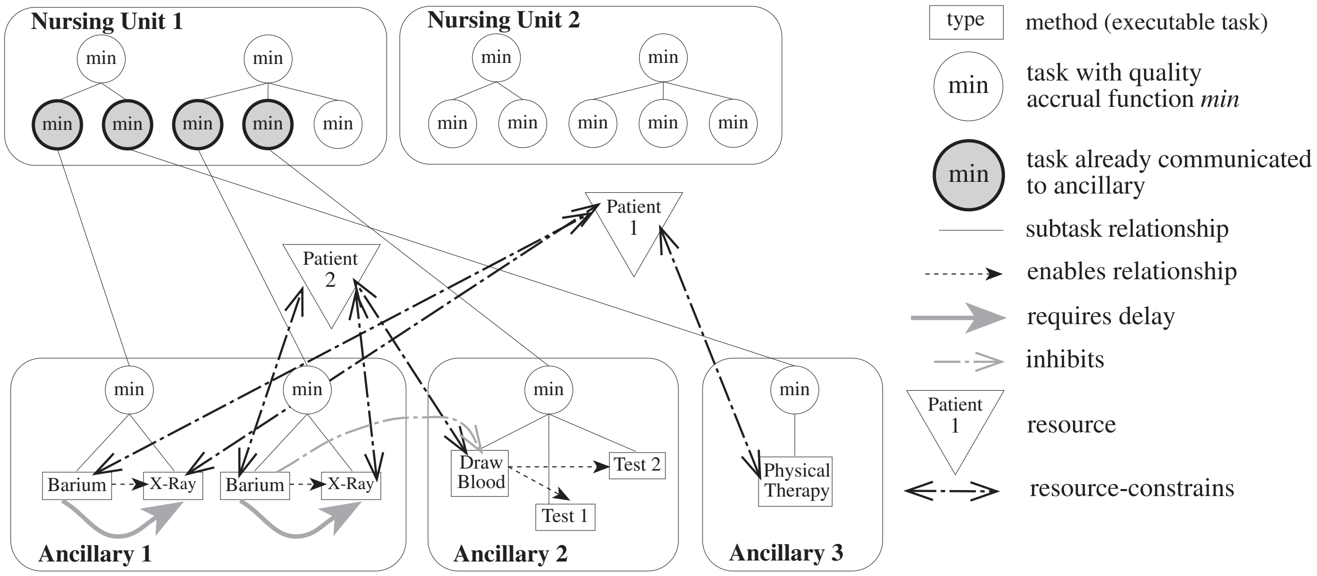

Figure 1. High-level, objective task structure and subjective agent views for a typical hospital patient scheduling episode.

> *Patients in General Hospital reside in* units *that are organized by branches of medicine, such as orthopedics or neurosurgery. Each day, physicians request certain tests and/or therapy to be performed as a part of the diagnosis and treatment of a patient. [...] Tests are performed by separate, independent, and distally located* ancillary departments *in the hospital. The radiology department, for example, provides X-ray services and may receive requests from a number of different units in the hospital.*

Furthermore, each test may interact with other tests in relationships such as enables, requires − delay (a slight variation on enables), and inhibits (a negative variation of the soft facilitates relationship). These task relationships indicate when the execution of one task changes the characteristics (here, primarily duration) of another task [7].

From this view of task structure, the hospital scheduling problem has these peculiar features:

- Tasks have no redundancy. Each test can only be done by a single ancillary department.
- The quality accumulation functions of non-executable tasks are always *min*. This is because here all the tests need to be done (TÆMS can however represent a wide range of quality accumulation functions).
- Unlike our previous domain models, here the final quality is not important. A test is either completed, or it is not.
- Different agents (representing different nursing or ancillary units) may use different utility functions when making local schedules. For example, the unit nurses may try to minimize the patient's stay time, while the ancillary staff tries to maximize equipment use and/or minimize setup times.
- Although TÆMS and GPGP handle them, there are no deadlines in this domain model.

Examining the hospital's current coordination structure is enlightening because it shows a mismatch between the structure and the current hospital environment. The current practice is for the nursing units to notify ancillaries of the requested tests and treatments, and for all ancillaries to schedule independently, sending an orderly to collect a patient whenever the scheduled time arrives. Of course, with no inter-ancillary coordination, there is no way to know if the patient will actually be there—they might be off having a different test performed. Unit nurses try to make sure the proper prerequisite tests (represented here by enablement constraints) are done first. While this structure seems sorely lacking when compared to the current environment, it may at one time have been a reasonable, low overhead arrangement. It may be that in the past doctors ordered fewer tests on less complex ancillary equipment with fewer or no interfering relationships between these technologies. Thus the current practice is adapted for a different task environment. In Section 5, we demonstrate that a simple algorithm, not much different from the current hospital scheduling practice, is not significantly different from our new algorithm when ancillary interrelationships are low or non-existent.

3. Generalized Partial Global Planning (GPGP)

GPGP is a domain independent *scheduling* coordination approach [1] The GPGP approach makes several architectural assumptions. Most important of these is that the agent represents its current set of intended tasks using the TÆMS task structure representation language [7]. These task structures are abstraction hierarchies whose leaves are instantiated basic actions or "executable methods". TÆMS allows the specification of dynamically changing and uncertain task characteristics that effect an agent's preferences for some state of the world, including tasks with hard or soft deadlines. A TÆMS specification also indicates relationships between local and non-local tasks or resources that effect these agent preference characteristics. An agent using the GPGP approach provides a planner or plan retriever to create task structures that attempt to achieve agent goals, and a scheduler that attempts to maximize utility via the choice and temporal location of basic actions in the task structure. Each GPGP mechanism examines the changing task structure for certain situations, such as the appearance of a particular class of task relationship, and responds by making local and non-local *commitments* to tasks, possibly creating new communication actions to transmit commitments or partial task structure information to other agents. The set of coordination mechanisms is extendible, and any subset or all of which can be used in response to a particular task environment situation. Initially, GPGP defined the following five coordination mechanisms based on Durfee's PGP:

1. Updating non-local viewpoints. Each agent detects the possible coordination relationships then communicates the related task structures.

2. Communicate results when they will be used by others.

3. Handling simple redundancy. When more than one agent wants to execute a redundant method, one agent is randomly chosen to execute it and send the result to the other interested agents.

4. Handling hard relationships (A must come before B) from the predecessor side. A is the "predecessor" task, B is the "successor".

5. Handling soft relationships from the predecessor side (if A is executed before B, the execution of B will be perhaps faster or will return better results, but it is not strictly necessary).

[1] The term "planning" in the name is historical, from Durfee's PGP. In the modern AI view of a continuum between planning and scheduling, both GPGP and PGP focus on the scheduling side.

4. Applying GPGP to Hospital Scheduling

4.1. The Minimal Required Coordination Mechanisms

To achieve the goal of finishing all the tests, some coordination mechanisms are required. One is the communication of results. To know which tests to do, and to whom to communicate the results, the agent needs to know the corresponding part of the task structure, so the mechanism of updating non-local viewpoints is also needed. Since there are no redundant tasks, we do not need the mechanism for handling redundancy. Handling hard and soft relationships is not absolutely required, since an agent can just wait around until it gets the needed results, and then begin the successor task (and in fact, this is slightly better than current practice). However, we will show later that coordinating over these relationships can improve performance. So, the simplest agent in handling the hospital scheduling problem will have just two coordination mechanisms.

4.2. A New Coordination Mechanism for Resource Constrained Problems

In the hospital scheduling problem, some (but not all) tests can only be done with the patient physically being there. Thus a resource constraint problem arises, where the patient becomes a crucial non-sharable resource for different agents. A mechanism that can responds to this situation is not included in the five mechanisms initially defined based on PGP, and can not be created by simply combining some of those mechanisms. Therefore we describe a new resource-constraint coordination mechanism. The new mechanism uses a simple multi-round but not multi-stage negotiation process which is not optimal but has good "flow" properties since at least one agent is free to stop meta-level communication and begin domain work at each round [12]. Our experience is that in highly dynamic environments with large amounts of uncertainty, excessive time taken to find an optimal or near-optimal schedule is wasted [5]. Other assumptions that we make include that the agents are essentially cooperating with one another, and that their utility measures are comparable (here, we are usually dealing with the amount of time saved, which is a comparable measure).

When several agents try to use the same non-sharable resource at overlapping times, only one agent can actually get the resource and execute its work. The others who failed to get the resource waste this time unit and this effort. The idea behind the resource-constraint coordination mechanism is that when an agent intends to execute a resource-constrained task (i.e. the task is scheduled), it sends a directed bid of the time interval it needs and the local priority (expressed as the effect on local utility) of its task (we'll describe how this is computed later). After a communication delay, it knows all the bids given out by the other agents at the same time as its own bid. Since all the agents who bid will have the same information, if they all use the same commonly accepted rule to decide who will get the time interval, they can get the same result on this round of bidding. The agent who won will keep its schedule and execute that task at the time interval it bid, and everyone else will mark this time interval with a *DON'T* commitment and never try to execute a related resource constrained task in it unless the owner gives it up. All the agents who didn't get their time intervals at this round will reschedule and bid again. The detailed process is as follows:

1. Get information about the bids given at the previous time unit, find who gets which time interval (i.e., the bid with highest priority), and make DO or DON'T commitments as appropriate. Also process retractions of commitments from agents that are giving up some time interval.

2. Reschedule. If the locally constrained task cannot be scheduled now, back to step 1. Otherwise, send out a new bid about the time desired and the corresponding utility.

3. Get information about the bids sent by others at the previous time unit, compare them to see if we get the time, and if not, mark the time taken by others and return to step 2. If we do get it, we also mark the time taken by others, and go to step 4.

4. Monitor the execution of the action in question. When we receive others' bids, we still mark the time intervals with the appropriate DON'T commitments for the scheduler just in case something goes wrong, the world changes, and we need to reschedule. If we find the task has significantly changed its execution time in the schedule, tell the others we give up the remaining time interval. If the task is taking longer, then go back to step 2 and bid again. If the task is finished, stop this process.

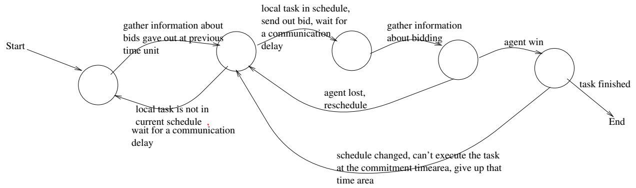

Figure 2. Finite state machine for the bid process

For example, let the communication delay be one time unit, and let there be three agents and tasks as follows: Agent A has task A11 of duration 3 and task A12 duration 3; Agent B has task B11 of duration 2; Agent C has task C11 of duration 4, and task C12 duration 1. A11, B11 and C11 have a shared resource-constrained relationship.

time 0: each agent communicates task structures.

time 1: each agent makes its own local schedule.

schedule: Agent A: A11(1-3), A12(4-6); agent B: B11(1-2); agent C: C12(1-1). Notice that task C11 is not in the local schedule of C, the reason may be some external causes, such as C11 not being enabled. The decisions about each schedule are made locally.

Agent A sends out a bid for time interval 1-3 with priority 3. (The agent decides the priority of a task according to its local view, it might be incorrect).

Agent B sends out bid for time interval 1-2 with priority 4.

time 2: each agent gathers information about the bidding at time 1.

Agent A finds that the other agent won time interval 1-2, so it marks this time interval occupied and then tries to reschedule. The new schedule is A12(2- 4), A11(5-7). Agent A sends out a bid for 5-7 with priority 3.

Agent B finds it won, so it keeps its schedule and starts its execution of B11.

Agent C puts C11 in schedule, C11(3-6) and executes C12. C sends out bid for 3-6 with priority 2.

time 3: Agent A finds that it won, so it keeps its schedule and begins to execute A12.

Agent C finds that it lost, so it marks time interval 5-7 occupied and reschedules. The new schedule is C11(7-10). C sends out a bid for time interval 7-10.

time 4: Agent C won, it keeps its schedule. C will wait until time 7 to begin its execution.

To put this mechanism to practical use, we need to decide two more things. One is how an agent computes the priority about its local task. The other is how to decide if an agent wins or loses according to the priorities of all the agents.

When using this mechanism to handle resource constrained problems, we will assume that each agent will be honest, i.e.. they will not cheat on the priority calculation. Since different agents may have different goals with respect to each other and with any global goals, the performance of such a system can be worse than that of a centralized system. In many domains such as hospital scheduling (or telescope observation scheduling [1]), We *cannot* centralize scheduling because it would take away the authority of each unit over the day-to-day control of its own activities. Our distributed approach matches with the existing human organizational structure. It also allows each unit to attempt to optimize slightly different measures, as may be used by administrators to evaluate human unit-level performance. In our work, we try to find the best priority function for agents that can minimize the finish time of the whole task. We found that (a) the coordinated relationships with the task and (b) the start time of that task should both be considered when computing the priority of a task. The reason for considering the coordinated relationships is apparent: a task that enables or facilitates many other tasks should have higher priority. The following example (Figure 3) will show why a task starting earlier should have higher priority.

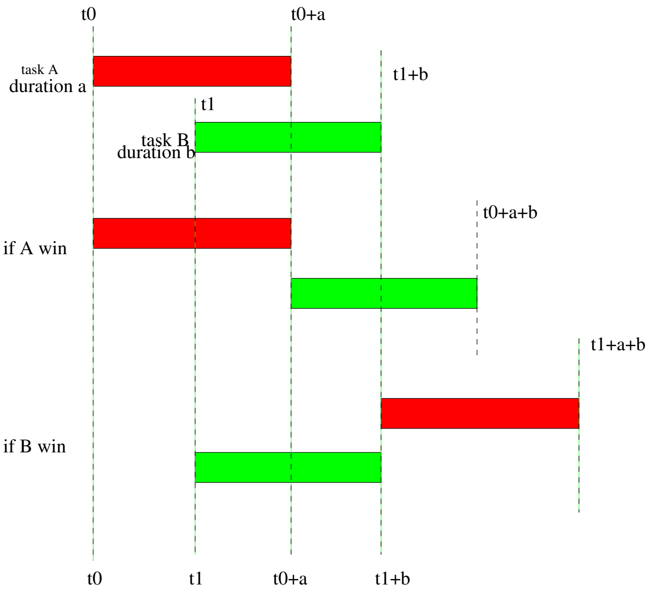

Figure 3. Why earlier task has higher priority

Assume task A and B start at $t0$ and $t1$ respectively, and $t0 < t1$. If A wins, both tasks can finish at time $t0 + a + b$, if B wins, the finish time is $t1 + a + b$. The reason is that the time interval between $t0$ and $t1$ is wasted. So, all other things being equal, an early starting task should have higher priority.

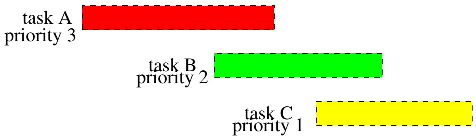

Figure 4. Example

It's easy to know who gets the time interval they want if there are only two tasks to compete for the resource. How about when there are more than two tasks? For example see Figure 4. Since task A has highest priority, it will win. But how about task C? Since task B can't win its time interval anyway(because of its conflict with A), there will be no conflict for C to be executed at its time interval, should task C win too? The results of those two different methods are shown in Figure 5.

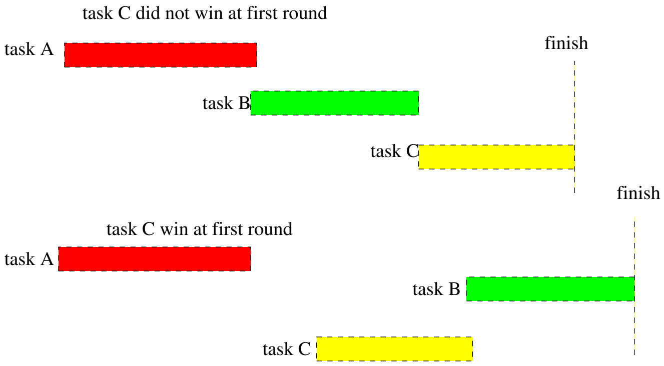

Figure 5. Result of different method to deal with multi-conflict

There will be two problems that occur when we let task C also win its time interval. First, task B actually has higher priority than C, but it lost its competition. In this situation, if a task lost in one round, it may have to start its execution very late even though it has a very high priority. Second, the time after task A finishes and before task C starts is wasted: nobody uses this time interval. The whole task will be finished late.

The advantage of allowing more than one task to get its time interval is it can reduce the rounds needed for solving the problem. When the cost of communication or rescheduling is high compared with the cost of execution, this method will get better performance. In our experiments, we used the first method where only one task wins at each round.

In domains other than this one, the priority calculation must take into account other characteristics over which agents may express preferences (usually represented as a utility function). One of the features of TÆMS is the indication of these utility-affecting changes in a precise, quantitative manner. However, since a peculiarity of this problem is that all tests must be done, reasoning about cost and benefit tradeoffs is not discussed here (but see [5]).

5. Experimental results

Since final quality is not so important in the hospital scheduling problem (tasks are either done or not done), we use the average task finish time and the actual work done by all agents to measure the performance of each agent. The task finish time shows us how quickly a patient can finish all the tests, i.e. the latency of the system. The actual work done by all agents can be used to determine the throughput of the whole system.

Our generative model of this environment focuses on the number and distribution of patient tests. The episode generator for Ow's hospital has a fixed number of nursing units and ancillaries, and a fixed set of test templates that correspond to taking x-rays, physical therapy, blood tests, etc. These templates contain statistical duration distributions on how long it takes to complete each action. Then, we postulate for each patient a distribution of tests (instantiated from the set of templates). Thus when each patient arrives, linked to that patient is a unique task structure of what needs to be done to that patient. In other words, to simulate the process of hospital scheduling, first we create a big task tree which represents all the possible ancillaries, tests and relationships between tests (figure 6). Note that some, but not all of the tasks require the presence of the patient.

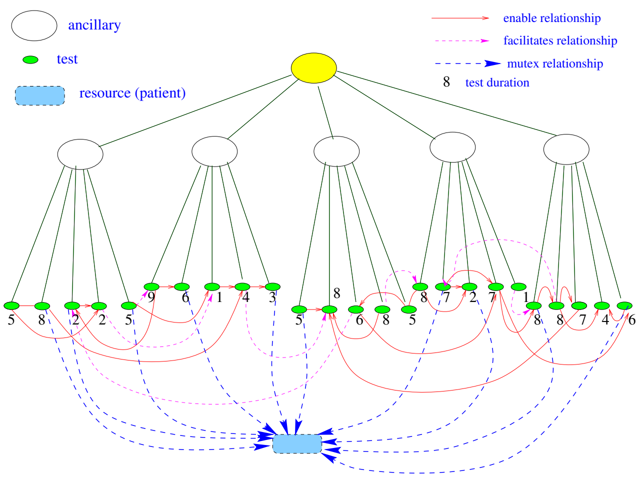

Figure 6. Task tree used in experiments

For each simulated patient, we stochastically choose ancillaries and tests from this big task tree. We use three independent variables to control the probability of non- local relationships (between ancillaries), local relationships (within an ancillary) and resource constraint relationships (how many tests require the patient to be present). Those variables have values between 0 and 1. A possibility equal to 1 means it will definitely contain all the relationships from the big task tree (but remember that this is *not* "all possible relationships").

All of the tests are done with an average of 3 ancillaries and an average of 3 tests in each ancillary. With each set of given possibilities, we run 40 repetitions. We compute the average difference in finish time of those tests. Given the hypotheses that the cooperating agents use *more* time than the simple agents, we use the Wilcoxon matched-pairs signed-ranks test to calculate the probability with which we can reject that hypotheses, i.e. show that the coordinating agents actually use *less* time. We also compute the average difference in actual work done by the two groups of agents.

From those results, we can show:

- The average finish time of cooperating agents is always less than that of simple agents.
- With an increase in non-local relationships, the finish time performance of cooperating agents first decreases then increases with respect to that of simple agents. The reason is because an agent has to use more effort to deal with the non- local relationships while it can not always get enough benefit from this cooperation. Our set of mechanisms is significantly better than just the two simplest mechanisms at the 0.05 level for structures with 80% or more of the actual number of inter-ancillary relationships.
- Performance in finish time of cooperating agents increases with the increase of patient resource constraints. Our mechanisms are again significantly better at 80% of the actual number of relationships.
- Throughput performance of cooperating agents increases with the increase of resource constraint relationships.
- The change in non-local relationships has no apparent affect on throughput performance; our mechanisms have a consistent advantage at all levels.

Each of the following figures has two graphs: the amount of time saved on the left, and the amount of work saved on the right. Higher numbers are better from the standpoint of our set of coordination mechanisms.

6. Related Work

Liu & Sycara[13] developed a coordination mechanism called Constraint Partition and Coordinated Reaction (CP&CR). In this mechanism, each resource is assigned to a resource agent which is responsible for enforcing capacity constraints on the resource, and each job is assigned to a job agent which is responsible for enforcing temporal precedence (i.e. *enables* in TÆMS) and release data constraints (like the early-start-time commitments in TÆMS) with the job. The job agents are similar to the agents in this paper, responsible for finishing their own jobs. They later extend the CP&CR to a new coordination mechanism Anchor&Ascend to deal with the Constraint Optimization Problem [14]. This mechanism also assigns resources to a resource agent. It identifies a bottleneck resource, and identifies the agent that is responsible for this resource as

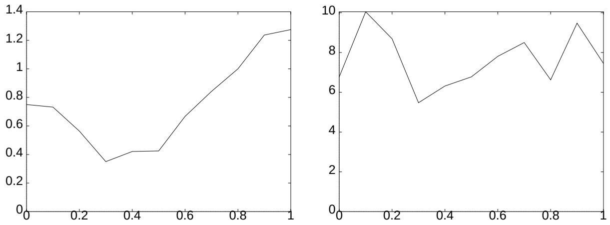

Figure 7. less time and work: increasing probability of inter-ancillary relationships (all patient resource relationships present)

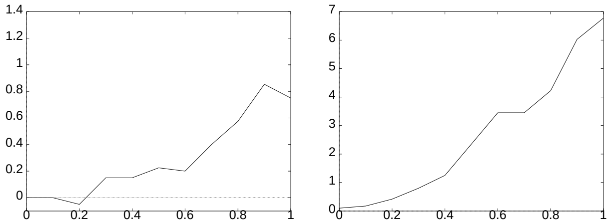

Figure 8. less time and work: increasing number of patient resource relationships (no inter-ancillary relationships)

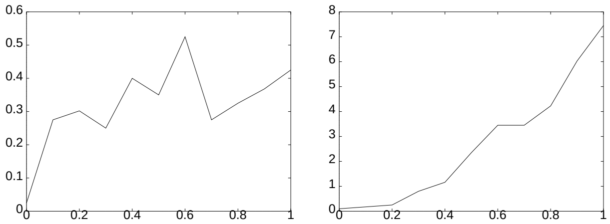

Figure 9. less time and work:increasing number of patient resource relationships (50% chance of standard inter-ancillary relationships)

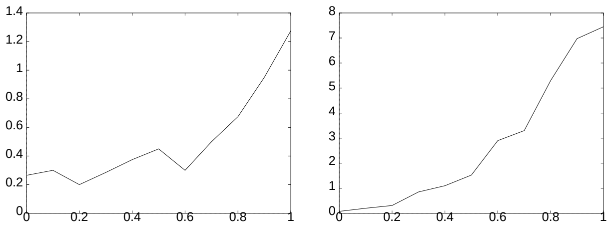

Figure 10. less time and work: increasing number of patient resource relationships (all inter-ancillary relationships are always present)

the Anchor agent. The Anchor agent makes the best local subsolution first, then all other agents make their own subsolutions while trying not to violate the subsolution of Anchor agent. The difference between our mechanisms and those two mechanisms is that CP&CR and Anchor&Ascend are partly centralized with respect to each resource. When dealing with hospital scheduling problem, the disadvantage of that style of solution is that it lets an agent outside of an ancillary to directly affect the ancillary's local schedule.

Ow, Prietula, and Hsu also develop a organization structure for hospital scheduling problem [15]. Their organization contains unit agents which are responsible for monitoring the test schedule of each patient as well as collecting and disseminating test requests to the ancillary systems, and the ancillary agent which is responsible for collecting the test requests sent to it and schedules them. We have used this same organizational structure. Ow's paper doesn't give any details about how each ancillary agent deals with the resource constraint problem, only that each ancillary agent will broadcast the times that have been blocked out to all other ancillaries (somewhat similar to the first step of the mechanism we propose here). Ow's mechanism also does not explicitly handle interactions between tests at different ancillaries. The set of GPGP coordination mechanisms described here easily respond to these inter-ancillary interactions as well as eliminate the full broadcast communication.

A great deal of interesting work has also been done on multi-agent meeting scheduling [17, 10]. This work has focussed on coordinating the resources and typically has not dealt with multiple hard and soft task interrelationships.

7. Conclusions and Future Work

In this paper we presented a model of the Hospital Scheduling Problem in TÆMS, and applied the GPGP approach to solving this problem. Hospital scheduling is interesting not only because it is a real problem, but because it is an inherently distributed problem, not amenable to a centralized solution because of the human organizational authorities involved. An extremely similar problem is that of scheduling telescopic observations at university observatories in the U.S. (each university wishes final control over the schedule of their own telescope, although they totally cooperate with other universities in scheduling observations)[1]. Some of the unique features of this particular problem that are very different from our previous work is that all the tasks need to be done, and the state of being "done" is either yes or no, rather than a set of continuous characteristics. There are no deadlines.

We did not represent travel times (and neither do the real hospitals), however, there is an implication of this downplay of travel times—the hospital will have to change its coordination and scheduling mechanisms if it has to face significant travel delays with *more than one ancillary*. This could happen if, for example, it is a smaller hospital and cannot afford its own MR scanner or CAT scanner and doctors take to prescribing MR and CAT scans at another hospital. Patients requiring *both* MR and CAT scans now have a significant facilitates effect in their task structures to do both scans nearly consecutively (potentially affecting the coordination structures of *both* hospitals). Other than changing the *representation*, we could handle this situation without changing the existing set of coordination mechanisms. We are currently looking to increase the model's realism by using more detailed data from real hospitals.

This paper also demonstrated the use of the GPGP approach in solving this problem. In this environment, the mechanisms to communicate task structures and partial results are necessary, while the mechanisms of handling hard and soft relationships are optional. By providing a solution without changing any existing mechanisms, we demonstrated again the generality of the GPGP approach. By limiting ourselves to just the two necessary mechanisms, we were able to model current practice fairly closely.

We then focussed on the specification and implementation of a new GPGP coordination mechanism oriented toward handling resource constraint relationships between tasks at different agents. The approach was that of a simple single stage, multi round cooperative negotiation, which has good flow properties and low overhead, which is important in an environment where the task mix may change dynamically with new patients, and where there is uncertainty over exactly how long procedures will take. We are aware of the utility of more complex multi-stage negotiations (e.g. [4]), and intend to examine the utility of such higher-overhead, higher-payoff approaches. The new mechanism we have developed can now be applied to any environment with resource constraints. Finally, we demonstrated the effect of the new coorination mechanism on performance in different environments. The experimental results show that the new mechanism increases the performance of agents by both decreasing patient stays and increasing throughput when there are many inter-ancillary relationships.

References

[1] J. Bresina, W. Edgington, K. Swanson, and M. Drummond. Operational closed-loop observation scheduling and execution. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

[2] C. Castelfranchi. Commitments:from individual intentions to groups and organizations. In M. Prietula, editor, *AI and theories of groups & organizations: Conceptual and Empirical Research.* AAAI Workshop, 1993. Working Notes.

[3] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[4] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. Systems, Man, & Cyb.*, 21(6), Nov. 1991.

[5] K. S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995. `http://dis.cs.umass.edu/~decker/thesis.html`.

[6] K. S. Decker and V. R. Lesser. An approach to analyzing the need for meta-level communication. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 360–366, Chambéry, France, Aug. 1993.

[7] K. S. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[8] K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, San Francisco, June 1995.

[9] E. H. Durfee and V. R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Aug. 1987.

[10] L. Garrido and K. Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the 2nd International Conference on Multi-Agent Systems*, Kyoto, June 1996.

[11] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[12] B. Lâassri, H. Lâassri, and V. R. Lesser. Negotiation and its role in cooperative distributed problem solving. In *Proceedings of the Tenth International Workshop on Distributed AI*, Texas, Oct. 1990. Also COINS TR–90–39.

[13] J. Liu and K. Sycara. Distributed problem solving through coordination in a society of agents. In *Proceedings of the Thirteenth International Workshop on Distributed AI*, pages 190–206, Seattle, WA, July 1994.

[14] J. Liu and K. Sycara. Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 246–253, San Francisco, June 1995.

[15] P. S. Ow, M. J. Prietula, and W. Hsu. Configuring knowledge-based systems to organizational structures: Issues and examples in multiple agent support. In L. F. Pau, J. Motiwalla, Y. H. Pao, and H. H. Teh, editors, *Expert Systems in Economics, Banking, and Management*, pages 309–318. North-Holland, Amsterdam, 1989.

[16] M. N. Prasad and V. Lesser. Learning situation-specific coordination in generalized partial global planning. In *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, Stanford, Mar. 1996.

[17] S. Sen and E. Durfee. Unsupervised surrogate agents and search bias change in flexible distributed scheduling. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 336–343, San Francisco, June 1995.

[18] T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.