A Real Time Multi-Agent Planner for a Specific Application

Carlos Pinto-Ferreira Alexandre Malheiro Bernardino César Santos Silva

Instituto de Sistemas e Robótica — Instituto Superior Técnico
Technical University of Lisbon
cpf@kappa.utl.ist.pt

Abstract

When applying planning approaches and formalisms to real-world problems two specific kinds of difficulties arise: on the one hand, the uncertainty related with system malfunctioning and changing in environmental conditions, and on the other, a *latency problem* that consists of not always having available a complete and correct description of the system state.
The paradigms of Hierarchical Planning and Interleaving Planning and Execution are applied successfully to a system belonging to the class of multi-agent loosely coupled systems in an changing environment.

Introduction

In this paper, a managing system for a real application, is presented. This system incorporates, among other modules, a planner and a housekeeper that illustrate the planning and learning features relevant to this presentation. The overall system, in which this managing system is incorporated, is a "public CD library" where users are supported by: (i) an expert system, helping them to choose musical themes, and (ii) a robotic system, carrying CDs from shelves to CD players and *vice-versa*[1]. The mechanical system is a star-shaped structure including five columns, each one containing 100 shelves and three CD players, allowing for the audition of up to 15 CDs by users, with a total storage capacity of 500 CDs. As each column is served by a gripper sliding vertically in a linear element positioned in front of it, the system is able to execute up to five user requests simultaneously. Some simulation tests were performed aiming at finding out among several topologies, the one providing the best tradeoff between cost and performance. In this case, the chosen configuration includes as mentioned, 500 CDs, 15 CD players, five columns and the corresponding linear elements. As there is the need of transferring CDs from one column to another (when all CD players in a column are being utilized and a user asks for a CD in this very same column), a carrousel located in the bottom of the structure, is also installed. In this case, the carrousel receives a CD delivered by a gripper of a linear element in a column and rotates to a position corresponding to another column, where another gripper will remove the disc. In order to connect each CD player with a user, a switching matrix is utilized.

[1]This system is somewhere in between a juke-box and an automatic warehouse.

When a user makes a request, which can be, for instance, listening a particular record, a management system should locate the corresponding CD and choose a CD player (in principle, the nearest with respect to the CD location). A planner should then establish an adequate course of action to achieve the goal of playing the CD, which includes defining a sequence of actions involving the affected agents (linear element(s), a player, the switching matrix, and, in certain cases, the carrousel). In what concerns current and pending request, the selection of the most adequate ones is critical to minimize the users average waiting time. Although there are some available "tricks" to reduce the perceived elapsed time (e.g., providing contextual information, pictures, drawings), in cases of severe system overload, if no effort is made to optimize the plan (for instance using a FIFO strategy), some users could have to wait for a long time.

Finally, an executor issues the corresponding set of commands in order to perform the planned actions. Although these commands are sequentially dispatched, the plan is a non-linear one whenever the carrousel is utilized, as the caroussel rotates and both linear elements involved travel simultaneously to ensure the transference at the carrousel. Albeit being a domain-dependent planner (allowing the incorporation of specific information and making things easier), the planner encompasses some degree of complexity, not only because it has to cope with multiple requests but also because real time decision making is demanded. Moreover, whenever a new request comes up, it should be integrated, so implying the need of replanning.

Furthermore, as the most frequently requested CDs should be placed near the players (for efficiency reasons) a statistic record should be maintained in order to permit the housekeeping of the system whenever the list of user requests is empty. The housekeeping facility should adapt to change in user preferences (e.g., from Mahler to Sinatra, from Gershwin to Armstrong), thus ensuring some degree of learning.

Motion and Housekeeping Manager

The Motion and Housekeeping Manager (MHM) is a module designed to dispatch, as efficiently as possible, the requests coming from the users and to ensure a convenient distribution of compact discs in the storage structure. This manager also provides an adequate utilization of resources, "disqualifying" over-requesting users.

The robotic system to be controlled includes, as mentioned, the following agents: five linear elements with the corresponding grippers, one carrousel, 15 CD players, and one switching matrix, which connects users to CD players. The MHM controls the behavior of agents by way of a series of commands sent over time. Each agent is able to interpret a set of commands, which are then executed as actions. All actions have a well-determined effect on the system state.

To achieve its goals, the MHM needs a great deal of information that must be incorporated as domain specific knowledge. This includes the knowledge about the set of available actions, the structure and the model of the system, and the way actions should be associated to perform tasks in order to satisfy users' requests. In addition, the MHM system can get information from its interaction with agents and users. It maintains an updated internal representation of the state of the agents and a database of discs' localization, lists of pending requests and statistic information needed to optimize the housekeeping task.

The MHM module is composed by two fundamental sub-modules: a *planner* and a *housekeeper*. The state of the system and the users pending requests are inputs to the planner that should derive the plan to be executed. The housekeeper uses statistical information and tries to maintain the CD's in adequate positions to allow for short waiting times. Other secondary sub-modules are included in the MHM system: a *state predictor*, which determines the state of the system in a certain future situation, according to the current plan, a *statistic processor*, that accounts for the frequency of requests for each disc, allowing for the adaptation of the system to the users preferences, and the *executor*, which receives the plan and provides commands to the agents (Fig. 1).

The Planner

In this section the planner sub-module is described and some details regarding relevant aspects of the implementation are addressed.

From the framework described above, it could be inferred that the planner interacts with a structured and well-defined world — so being possible to maintain a representation of it and to predict its state after a specified number of actions. However, in this application, there are two sources of unpredictability: system malfunctioning, (although unlikely, it must be taken into account), and the coming up of a new user request, which has a random character. As the planner ought to consider, for efficiency reasons, all the requests already sent by the users, a new plan should be derived whenever a new request comes up. Therefore, replanning should be performed frequently; however, this process — a very time consuming one — should not endanger real time functionality.

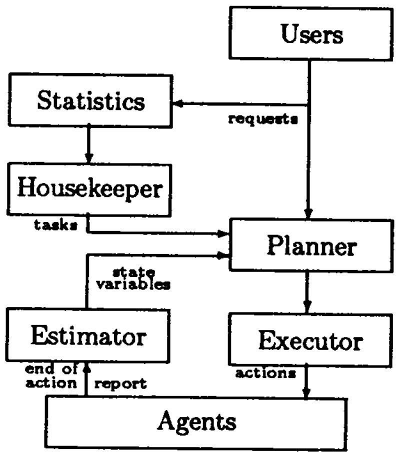

Fig. 1: Overall organization of the Motion and Housekeeping Manager.

To circumvent this problem, a pragmatic approach to planning was decided: instead of establishing a plan for all requests, the planner only handles the most adequate one.

This results in an interleaving plan formation and execution (Georgeff & Lansky 1990), avoiding the need of heavy replannings because no commitment is made on a complete course of action. Basically, the system forms a partial plan, acts accordingly, modifies the plan when a new request arrives or a task starts to be executed, executes further and so on.

To serve efficiently the user requests, the waiting period (the time elapsed from the instant requests are made until listening is started) should be minimized. Inspecting deeper the structure of the system, it can be seen that the linear elements do not interact directly. The interaction is limited when accessing the carrousel. Thus, the overall system can be considered as composed by five loosely coupled sub-systems (Stefik 1981). The interactions with the carrousel are initially abstracted and it is possible to linearize the plan and to obtain five independent sub-problems. The optimization performed to minimize the waiting time is easy when applied to linear plans. In fact, it can be proved that the plan is optimal in these conditions. Of course, when interactions arise, the plan becomes sub-optimal, but the interactions are the exception and not the rule.

As sub-problems can be isolated when taken in a higher level of abstraction, a hierarchical planner is designed and encompasses two sub-systems: a *high-level planner* and a *low-level planner*. In the former, some tricky aspects (such as the interactions among agents and the carrousel access and motion) are ignored. These aspects are considered as minor details, having little impact on the global optimization procedure, so not being considered in this high level sub-system. These aspects are included in the low level planning sub-system that ensures correct interaction among agents and local carrousel access and motion optimization.

High-level planning In the present application, to formulate complex plans for all situations is a waste of time, since frequent replanning may be needed. Instead, unimportant aspects to the global optimizing strategy are ignored. These aspects include the carrousel access and motion and the interaction among agents. Thus, agents are considered to execute actions sequentially and carrousel as being autonomous. Note that there is an interaction among several agents, but in a weak way. It is in this sense that the hierarchical planner approach as proposed above.

The High-Level Planning sub-module produces a plan to accomplish the goals associated with the incoming requests, by the way of a task level decomposition and processing. There are two fundamental tasks requiring planning: the user wishes to listen to a disc, and she/he wants to stop the audition. Each task has a set of parameters that defines its goal, which includes the position where to move to the intervening agents, defined by the position of the target disc and player.

The MHM system associates a single macro action with each task. A macro action is a set of actions directed to a *single* linear element and its corresponding workspace, including shelves and players. Then, a macro action is appended to the list of pending macro actions. There exist five lists of macro actions (one for each linear element) and each list corresponds to a single server (a linear element).

The following hypotheses are raised:

(i) the server executes macro actions sequentially;

(ii) a server has n macro actions to execute at an instant of time;

(iii) the execution time required by each macro action is known in advance: the macro action i will take time t_i, $1 \leq i \leq n$.

It is intended to minimize

$$T = \sum_{i=1}^{n} \tau_i \qquad (1)$$

where τ_i is the elapsed time between the instant of macro action creation and its execution.

Minimizing T is equivalent to minimizing the time spent to serve each user, if it is supposed that the carrousel is not utilized and a single macro action is enough to complete the user task (this is not always true).

In these circumstances, a greedy algorithm can be applied: at each step, add to the end of the list of macro actions, the macro action requiring the least execution time among those which remain. This algorithm is optimal if conditions (i), (ii), and (iii) are met as demonstrated in (Brassard & Bratley 1988). This strategy is followed in the present application.

The following items are steps of the algorithm that processes the lists of pending macro actions: (i) the adequate player and shelf — allowing the execution of the task — are selected, (ii) the execution time for each macro action is estimated, this time being affected by a priority factor, that corresponds to the user priority, (iii) the macro action requiring the least execution time is selected and it is inserted in the end of the list of pending macro actions, (iv) only the last macro action of the list is analized, (v) if all execution conditions are met, the executable macro action is removed from the list, and (vi) auxiliary macro actions are created (for instance, if a transference using the carrousel is needed). These steps are performed when: (i) a new macro action is introduced,(ii) the MHM system detects a change in the robotic system (for instance, an error), (iii) the user priority changes, and (iv) a macro action is transferred to the low-level planner.

The executable macro action requiring the least execution time is expanded into elementary actions, which are executed following a procedural net.

It should be pointed out that the algorithm is now sub-optimal; the conditions having changed as follows:

(i) the execution time required for each macro action is now dependent upon the order of macro action execution that changes dynamically;

(ii) users can be served by interacting agents;

(iii) t_i is now dependent of the estimated execution time of macro action i, which is affected by a priority factor.

However, the algorithm has the advantages of a greedy approach, namely: in each list of macro actions it is only necessary to know the macro action i having the least execution time and it is not necessary to sort or to instantiate the remaining macro actions.

Low-Level Planning In this sub-system, the problem of ensuring a correct interaction among agents is addressed and the issue of locally optimizing some aspects, not considered in the high-level procedure described in the last section, is approached.

As there are several agents to command, there exists a large degree of parallelism to deal with. When some of the agents interact with each other, there is the need of synchronizing their activity. In order to prevent conflicting situations, a set of precedence rules, for several actions was defined.

These precedence rules were implemented using a procedural net (Sacerdoti 1975; Tate 1990) where nodes represent actions and arcs represent the flow of activity. An action can only be executed when all actions included in its parent nodes have been performed; the MHM system is informed about the ending of an action as the executing agent reports the fact.

When building the net, actions directed to the same agent are sequentially inserted and when different agents conflict with each other, a precedence rule is applied. In this way, sequential execution for actions concerning an agent is ensured and disjoint execution for actions involving several agents is obtained.

The net only includes the actions that accomplish the current requests. All pending requests are kept in auxiliary lists to facilitate replanning. As was mentioned in the previous section, when a new request arises, replanning is done in a higher level, and does not interfere with the current requests' treatment.

A useful property of the net, with direct application in the planning process, is that it enables the knowledge of a future state of the system, determined by the end of the last action included in the net.

Some optimization aspects were left behind when the high-level planning sub-system was described. It is now time to consider these aspects. When expanding a macro action into the net, some decisions may have to be made: this happens when the task demands an access to the carrousel. The low-level planner has to determine the shelf to access and the angle of rotation needed. This is done by a search procedure that finds where the required disc is located or the nearest empty shelf among the shelves of the carrousel. This way, the local planning strategy becomes easy and a considerable amount of computation in the high-level planning sub-system is avoided.

Operational Problems

The end-of-action reports supplied by the agents when finishing an action allows the maintenance of an updated internal representation of the system state. This data is kept in state variables saved in the computer memory and represents what is called the *current state.* The current state corresponds to the actual state of the system when actions are not being executed.

Many routines used by the planner require access to the state variables. For example, to find the location of a particular disc in the structure, the routine searches through the state space. When agents are executing actions, the current state is not very useful. If a gripper is currently removing a disc from a shelf, an access to the current state variables reveals that the gripper is empty. However, if nothing goes wrong, some instants later, it will contain the removed disc. In fact, there are periods of time where the state is undefined. If actions frequently fail to be executed, either the planner could not plan ahead with a good degree of certainty or a branch should be created considering the two situations (success and failure in executing the action). This *latency problem*, often ignored in the design of planning systems to operate in simulated environments, where execution of actions is instantaneous, must be considered in real-world applications.

Fortunately, in most structured domains, failure is not common, permitting to assert that the consequences of the current action are met in the present instant. So, a *virtual state* is defined, corresponding to the state the system will be after all current actions have been executed. It is inferred very easily from the current state information and the current actions being performed.

For some purposes, neither the current state nor the virtual state provides the state information needed to optimize the plan. The high-level global and the low-level local optimizing routines need to know the state of the system when all actions in the net be performed, considering that they will be successfully executed. This is the *predicted state* and can be inferred from the current state and using procedural net information. Making use of these properties, it can be proved that it is possible to know most of the predicted state variables important to planning, through directed backtracking strategies. These include:

- the position of the linear elements and carrousel;
- the discs contained in the grippers and the shelves (including the carrousel shelves);
- the players connected to users, etc.

Having this information, the planner can predict the state of the system in a future instant, allowing a better global and local optimization.

The Housekeeper

When users are not requesting discs, housekeeping tasks can be executed. These tasks store the discs in the structure, providing better access times (time needed to reach a disc) and better delivery times (time needed to move from the disc position to the player position) when serving the users' requests.

This optimization feature makes use of statistical information. Users have musical preferences that can be estimated calculating the frequency of the requests with respect to each compact disc within a certain period of time. To each disc is associated a probability of being requested, thus providing a way to integrate some decisions in a probabilistic formalism. The set of discs defines a probability distribution along the corresponding column.

The average access time to a shelf containing a certain disc is minimized if the gripper is positioned at the expected value of the probability density function. This procedure is only feasible if the gripper is not in operation (not attending other requests).

The average delivery time is minimized if preferential discs are placed near the cd-players. This represents the function of the housekeeping module. It must detect when preferential discs are far away from the players and, if possible, place them in near locations.

The adaptability to users' preferences, using knowledge got from users' requests allows the improvement of the overall system performance.

Some Results

Several tests were performed in order to evaluate the performance of the system. Even exposed to very severe situations (intense flow of requests), it exhibited a very robust run-time behavior. In reduced load situations, its delivery time performance is similar to a FIFO approach (Fig.2), as it was expected (in this figure, horizontal axis represents the average number of request per unit of time ($p = \frac{1}{32}$), and vertical axis represents the average waiting time). Requests are dispatched as they arrive to the MHM system and not often they are processed by the planner in the pending request list. However, it shows drastic improvements in the delivery time, when overload situations are considered. In fact, the waiting time of each client is lowered, compared with the FIFO algorithm. Note that in very hard situations the proposed algorithm serves clients who give up in the FIFO case — this modifies the average waiting time — asterisk (*) in Fig.2.

The adaptability to users' preferences results in an additional improvement in the system performance. This fact is more relevant in situations with reduced flow of requests, because the gripper has time to get to the best position. In other circumstances, the agent is too busy to perform this optimization.

Finally, there are some improvements to be implemented in the system, namely concerning the adaptation feature described above. In the phase of choosing the record to listen to, users interact with an expert system that incorporates the knowledge provided by a group of experts in several areas of music. In this phase, it is possible to detect users' preferences and to predict — with some degree of uncertainty — the user final choice, so allowing to position a gripper in a place in order to reduce delivery time.

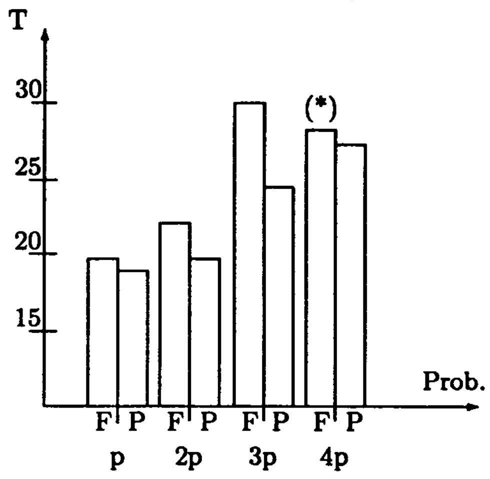

Fig. 2: FIFO (F) and proposed (P) algorithms.

Conclusions and Future Work

Notwithstanding being a particular application, the reported planning approach can be extended to cope with other systems having similar characteristics. The paradigms of Hierarchical Planning and Interleaving Planning and Execution were applied successfully to a system belonging to the class of multi-agent loosely coupled systems in an environment where unpredictability is frequent. Besides having found that these planning paradigms are well suited to this class of problems, the possibility of linearizing the plan leaving the nonlinear features to a lower level of abstraction, has permitted the successful application of optimizing procedures. The ideas described in the housekeeping module (the adaptability to user preferences) where developed having in mind this specific application; however, in a more general point of view, statistical information, when available, can be used to adapt systems to the environmental circumstances.

References

Brassard, G., and Bratley, P. 1988. *Algorithmics, Theory and Practice.* Prentice Hall.

Georgeff, M., and Lansky, A. 1990. Reactive Reasoning and Planning. In J. Allen, J. Hendler, A. T., ed., *Readings in Planning.* Morgan Kaufmann Publishers, Inc. chapter 11, 729–734.

Sacerdoti, E. 1975. "The Nonlinear Nature of Plans". In *Proc. of the Fourth International Joint Conference on AI,* 206–214.

Stefik, M. 1981. "Planning with Constraints". *Artificial Intelligence* 16:111–140.

Tate, A. 1990. Generating Project Networks. In J. Allen, J. Hendler, A. T., ed., *Readings in Planning.* Morgan Kaufmann Publishers, Inc. chapter 5, 291–296.