Incremental Mission Allocation to a Large Team of Robots

Thierry Vidal, Malik Ghallab, Rachid Alami

Incremental Mission Allocation to a Large Team of Robots

Thierry Vidal Malik Ghallab Rachid Alami
LAAS-CNRS - 7, av du Colonel-Roche - 31077 TOULOUSE - FRANCE
e-mail: [thierry, malik, rachid]@laas.fr

Abstract

In the MARTHA project, a large number of robots in a harbour are given the global task of transporting containers from one area to another. The global decision-making process of allocating robots to those predefined tasks can be viewed as a scheduling and resource allocation problem, which is addressed here in a centralised way.

Imprecision of temporal constraints make it meaningless to search for a strict optimal schedule. Our approach interleaves task allocation and execution, scheduling in a sliding short-term horizon, as the execution process runs, and providing near-optimal solutions.

For largeapplications as our, the complexity of temporal management is a crucial issue. We present here a graph decomposition technique, leading to nearly-constant time temporal propagation, without any loss of information.

1. Introduction and Related Work

We consider the task where a large number of robots have to carry containers from place to place (boats, trains, planes, stocking areas) in a harbour or an airport. The planning and execution of routes are distributed along the robots, with resource sharing techniques (crossroads, parking areas, ...) through a plan merging operation paradigm (see [1] for details). But this process has to rely on a global mission allocation system, whose description is the purpose of this paper.

Deciding which container will have to be managed next, and which robot will carry it, which is a classical scheduling and resource allocation process, is constrained by temporal optimality criteria (containers have to be unloaded from boats and trains before their departure times), which requires the use of heuristic functions along a greedy search approach. Moreover, in our application context, we are mainly concerned with the imprecision of temporal constraints on expected events and goals. For example the actual duration of an action depends upon the execution level and the actual state of the environment. Short term predictions and objectives are then usually quite reliable whereas long term ones are not (what [2] calls the «scheduling uncertainty»). This lead us to adopt a dynamic approach, interleaving mission allocation and execution, as in [5], or in [6]. So, we will make local decisions, in order to incrementally build the solution along with its execution, taking into account the current results of the process going on.

Therefore, the efficiency of the allocation process becomes a crucial issue. Like in [7] or in [4], the temporal constraints propagation process is at the heart of the problem, making explicit new constraints that will help guide the scheduling choices, which will in turn add new constraints that will need to be propagated. But [7] as most authors use costly global propagation algorithms, whereas [4] uses focusing techniques that, although enhancing the propagation efficiency, do not assure its completeness any longer, and then leads to «deviations» in the decision process.

Our proposal is to decompose the global temporal graph into sub-parts (or clusters) in which complete propagation can be confined. Our decomposition scheme relies upon the application-dependent structure of the graph, whereas for example [6] proposes a temporal decomposition based upon complex problem characterisation techniques. We then get a propagation process that is fast (nearly constant time), complete and gives sound answers to the queries made by the decision-making loop. Such a temporal system combining expressiveness, robustness, and efficiency, dedicated to large real problems, constitutes our major contribution.

Moreover, [8] and [2] tell us that dynamic systems cannot guarantee the quality of the final solution, regarding some global criteria, and that only near-optimal solutions can be reached. We show that in most of the cases we had to study, the proposed system found solutions of high-quality.

Part 2 will present the robotic application context and the type of problem addressed, part 3 will focus on the temporal management aspects, then the allocation process will be depicted in part 4 and the execution process, together with the overall interleaving process, in part 5. We will conclude following experimental results given in part 6.

2. Application Context and Type of Problem

The application context, inspired by the MARTHA project described in [1], is depicted in Figure 1. In a high-level view, the envrionment (let's say a harbour) is decomposed

into virtual areas. We only know the imprecise duration needed to go from one area to another. Containers (a few hundreds per mission) arrive in boats or trains (we will use the generic term of «conveyor»), which dates of arrival and departure are also imprecise. They are to be brought to some other place: stockage areas, or conveyor areas, which defines a set of «containers-shifting» missions that are to be carried out by a crew of robots (about 50).

The generic mission performed by a robot is a sequence of the four actions :

- Moving from the robot current position to the container loading area (GOTO action).
- Picking up the assigned container by use of a crane located in this area (PICKUP).
- Moving to the unloading area (GOTO).
- Putting down the container (PUTDOWN).

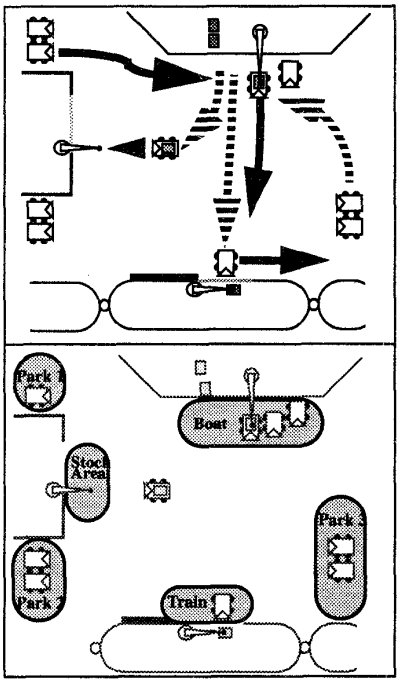

FIGURE 1. Illustration of the application context

Moreover, the PICKUP and PUTDOWN actions have to be made during the availability of the corresponding conveyor in the area, i.e. between its arrival time and its departure time. This defines an initial temporal window during which the PICKUP and PUTDOWN actions have to be conducted.

The containers are attainable in a conveyor in a given partial order (which is represented through «virtual stacks» of containers in each conveyor area). This defines an initial partial ordering of PICKUP actions. On the opposite, the ordering of PUTDOWN actions is not constrained in any way. There is just one crane per area, which is implicitly managed throughout the strict ordering of PICKUP/ PUTDOWN actions in each area.

As far as the type of problem is concerned, we can see that we have to face what [11] calls a joint problem, mixing scheduling and resource allocation. The first step (see part 4) consists of iteratively choosing a mission (i.e. a container) and the resource (i.e. the robot) that will be allocated to it. This is a simpler resource allocation problem than in [2] as for us the «robot» resources are all equivalent and can be equally allocated to any mission.

The second step addresses addition of strict ordering constraints
- between successive missions for each robot,
- between PICKUP/PUTDOWN actions on each area.

3. Temporal Management Issues

Incrementally adding new temporal constraints needs updating and consistency checking capabilities. For this purpose, our system relies on a complete and separated temporal constraint management system, which can be compared to [3] system relying on the Deans's TMM, and also to [7] in which resource constraints are used to add new temporal constraints.

3.1 Basic Representation Issues

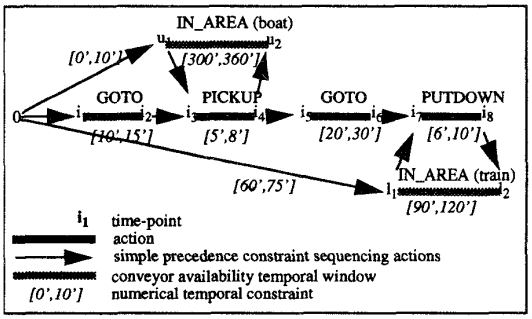

FIGURE 2. Representation of a mission-graph

The representation of time within our temporal system IxTeT relies upon a graph-based structure with time-points (the nodes in the graph) that are constrained by precedence constraints (directed edges in the graph), and also by imprecise numerical constraints (durations and dates) given as intervals of possible durations labelling the precedence edges (see details in [9]). This leads to represent the above generic mission through what will be called a «mission-

graph» in Figure 2, with the four elementary actions and the initial conveyors availability temporal windows.

At the beginning of the allocation loop, the global graph is expanded, with all the mission-graphs in parallel, not yet connected one to another. In Figure 3, one can see that each time a robot is allocated, precedence constraints corresponding to the interactions with other mission-graphs are added (doted arrows). Thus, the global graph evolves from a highly parallel one to a more sequencial one, where, as usually in scheduling problems, missions for a robot are incrementally ordered, and constraints between those robot sequences of missions appear because of the ordering of the PICKUP/PUTDOWN actions.

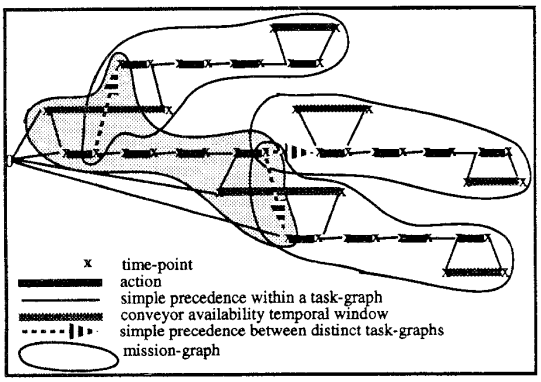

FIGURE 3. The decomposition scheme

As our application involves large number of robots and containers, we have to manage big dimension graphs, in the order of a few thousands time-points, and so classical global propagation algorithms are too expensive in our allocation/execution interleaving approach.

3.2 The Clusterised Propagation Technique

In order to improve the propagation complexity, we have put forward the following techniques.

Figure 3 reveals that each time a mission-graph is «instantiated» with a robot, a new time-point is also added to it: the one corresponding to the end of the previous mission allocated to the robot (the «availability time-point» of the robot). And the new PICKUP/PUTDOWN ordering constraints leads to the updating of the temporal windows for the PICKUP/PUTDOWN actions, which is represented in Figure 4.

This definition of the evolution of the mission-graphs during the incremental allocation process leads to the following property:

• The decomposition of the global graph into individual mission-graphs defines a complete partition, i.e.
- any time-point belongs to at least one cluster,
- any input constraint belongs to at least one cluster.

This basic property allows us to put forward the following recursive propagation process. Let us call C_u the mission-graph (or «cluster») corresponding to the transportation of the container $Cont_u$.

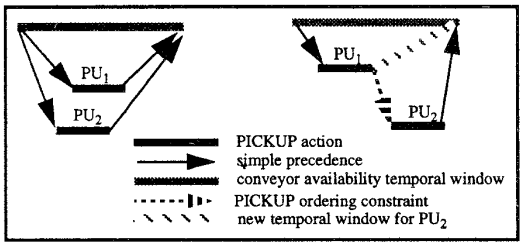

FIGURE 4. PICKUP temporal window updating

• **«Clusterised» propagation process**

If (i,j) is a temporal constraint being modified, we easily conclude from the previous definitions that necessarily $\exists$ C_u such that $i \in C_u$ and $j \in C_u$. Then we execute *PROPAG(i,j,C_u)* which is composed of the following steps:
(1) propagate (i,j) only within C_u,
(2) for each modified constraint (k,l) belonging to another cluster C_v, execute *PROPAG(k,l,C_v)*.

From this definition, we prove that:

(1) *The clusterised propagation is complete: it always detects an inconsistency if there is one.*

Moreover, the only temporal constraints that are useful for the global decision-making process, as we will see in next part, are:
- the «availability date» of a robot, i.e. the imprecise date of the «availability time-point».
- the temporal windows for the PICKUP/PUTDOWN actions.

Thanks again to the definition of the mission-graphs and of the clusterised propagation process, we prove that:

(2) *We get complete and accurate answers to queries made on the above mentioned temporal constraints.*

Thus we can say that our clusterised propagation process behaves in the same way as a global propagation would do, regarding the global decision-making process we have to address.

Concerning the efficiency, the decomposition leads to the definition of sub-graphs each containing 14 time-points. We thus get a complexity of $O(k.14^3)$ at each constraint addition, with k being the number of sub-graphs that will have to be propagated in a recursive way.

So we have put forward a much powerful, though complete, propagation algorithm, taking advantage of the particular structure of the temporal network in our application.

4. The Allocation Decision-Making Loop

The decision-making loop is made of iterative steps of allocation of one robot to one container, with the aim of not backtracking on these decisions (in a dynamic behaviour perspective). At each of these steps, we get as input:

(a)- the current overall graph, with mission-graphs already ordered (corresponding to missions already allocated) and unconnected mission-graphs (corresponding to the missions not yet allocated),

(b)- the current position and «availability date» of each robot (i.e. the resource availability),

(c)- the current virtual stacks of containers remaining on each conveyor, and the unloading area for each container (i.e. the set of missions requiring sequencing and resource allocation).

Then we have to run the two decision steps that are usually addressed in search techniques ([2]), especially when backtrack-free search is required:

1) «*variable ordering*»: which mission to schedule next, i.e. in our case which container to take next.

2) «*value ordering*»: which resource and temporal placement to choose for this mission, i.e. in our case which robot will take care of the container.

These choices will be made through heuristic functions depicted in section 4.1. This will lead into updating operations concerning data (a), (b) and (c), mainly adding temporal constraint (described in section 4.2), ending into a new situation.

4.1 Choice Strategies

As it is discussed in [2], «there is no such thing as an optimal schedule», because optimality criteria are usually multiple, fuzzy and conflictual. In our application domain, one temporal criteria (as in [10]) appeared to be pre-eminent: trying to process all the loading/unloading actions with a minimal loss of time, in order to get the best chances of having unloaded/loaded all the containers in a conveyor before its departure.

We thus want to get some near-optimal (or «good quality») solution according to this criteria. First, in an off-line initial process, we can check if the problem is globally feasible, simply checking if the sum of the durations of the loading/unloading actions on each conveyor is lower than the total duration time of availability of this conveyor.

Then, at each step of the allocation process, the two following heuristic functions are processed.

- **Find the «most critical» container.**

This requires knowing which are the «reachable» containers, i.e. the ones that are on top of the stacks. This is easily done thanks to a graph-based structure of these stacks.

To meet the global criteria defined above, we will have to look for the most urgent container, i.e. the one for which there is the less time remaining for catching it. This leads to search for the container whose latest possible date of loading it is minimal (i.e. search for $Cont_u$ that minimizes the upper-bound of the numerical constraint on the date of the starting time-point of the PICKUP action). This defines a first «criticality dimension» for each container.

From this definition, two containers loaded from the same conveyor will be equivalent. Thus we will need another choice function, reasoning on the unloading area: we will choose the container for which we won't need to wait before putting it down (e.g., if the container has to be unloaded on a train that arrives in 3 hours, it is not necessary catching it now as it will unnecessarily bring a robot to a standstill).

Of course, these functions might be improved in order to get better solutions in more complex situations. But this has been found to be satisfactory enough in practice, when addressing current situations in the harbour (see part 6).

The time complexity of this function, since numerical requests are made in constant time in the mission-graphs, is only linear in the number of reachable containers, which are generally few, as far as the containers are «highly stacked» in the conveyors.

- **Find the best robot.**

The best robot, again following the same global criteria, will obviously be the nearest to the loading area, i.e. the one that will arrive first. We then only need to compute the expected arrival of the robot *ExpArr* as the sum of the current availability date *AvD* of the robot and the goto duration for reaching the area, and choose the robot $Robot_i$ that minimizes this dimension.

Again, the function takes advantage of the propagated temporal constraints and its complexity is only linear on the number of robots.

4.2 Adding new Temporal Constraints

Once the choices have been made, temporal precedence constraints will be added (and propagated)

- between the last availability time-point of the robot and the beginning of its new mission,
- and between the PICKUP/PUTDOWN actions that have to be strictly ordered.

Those last ones are added in a least-commitment approach. Because of the choice criteria, (the container which has to be picked up first will be allocated first), there is no particular problem about the PICKUP actions: they are ordered as soon as they are allocated.

For the PUTDOWN actions, it is not as easy. Let us see the following situation of Figure 5, in which robot Rob1, in area A, is allocated to the most critical container Cont1, to convey it from area A to area C, which takes about 20 minutes. Then Rob2, in area B, is allocated to Cont2, which has to be taken from area B to area C, which takes about 5 minutes. Thus, Cont2, which has been allocated after Cont1, will arrive before ! The PUTDOWN of Cont2 should then be inserted before the PUTDOWN of Cont1 (as a need for optimality), which may delay it, and then change the current availability date of Rob1. In a backtrack-free search approach, we cannot allow to change it at any time.

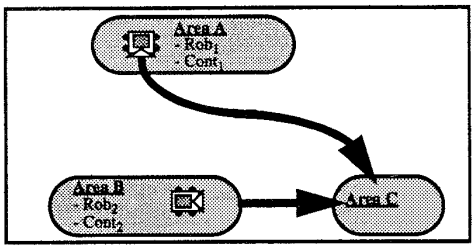

FIGURE 5. Conflictual example

Fortunately, we can:

1. observe that we only need to know the availability date of a robot when we allocate it to a new mission,
2. and then easily prove that we can delay the PUTDOWN action ordering until the robot is reallocated.

Let us see it in the previous example. If Rob1 is reallocated to a new mission, and if afterwards the situation in Figure 5 arises, we then get a contradiction: Rob2 would have been chosen for carrying Cont1.

The last situation is when a PICKUP and a PUTDOWN possibly overlap in a same area. Here, the ordering decision is made by a heuristic comparison of the criticality of the involved containers (see last section).

Thus, we have put forward a least-commitment approach, mixed with some heuristic choice when needed, to assume a backtrack-free near-optimal ordering of the loading/unloading actions throughout the global process.

5. The Execution Process and the Global Control Loop

As we have sketched out in the introduction, one cannot allocate the robots to all the missions, because of absence of precision in the numerical constraints given in input. The more missions we give to a robot, the more imprecision we get in its new availability date, because of imprecision growing within successive propagation processes. A point is reached when it is no more possible to choose between two robots in a sufficiently deterministic way: the availability dates of two robots being intervals of possible values, we are sure that one will arrive first if those two intervals are non overlapping. If there are overlapping, we measure the «overlapping ratio» *OvP*:

OvP = length (overlapping) / length (union of the two intervals)

When *OvP* exceeds some threshold *T* given as input, then this overlapping becomes a dead-lock as we can only make an unreliable choice that could lead to a future need to backtrack. We then decide to stop the allocation loop and wait until the execution process gives account of more precise values that will reduce the dimension OvP: each executed action brings some real duration value that is propagated to the following expected actions, thanks to a directed arc-consistency algorithm that runs in $O(m.r)$ in worst case, where m is the number of time-points in the graph managed by the execution process, and r the number of robots.

The question is now: how will we fix the dimension *T* ? Obviously, if we have a lot of imprecision, we cannot require a tight discrimination criteria between two robots (formally a short overlapping ratio), because we may not be able to make choices and the robots could remain unallocated. Then the global system would stop. So, we need to build a function that computes the correct threshold *T* depending upon the imprecision of the temporal numerical constraints given in input, such that each robot is given at each time an horizon of one up to two missions.

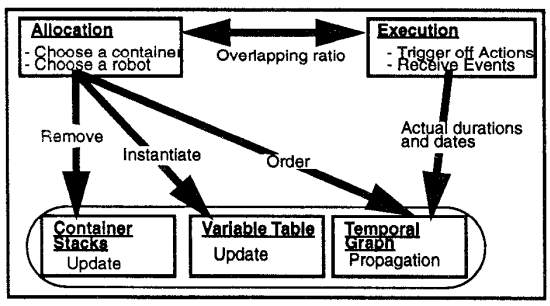

FIGURE 6. Global architecture

As we only allocate within a short-term horizon, the total number of mission-graphs corresponding to the missions being already allocated, but not yet executed (the dimension k in part 3), is strictly bounded, which leads to nearly constant time propagation process. The described approach leads to the global architecture of our system, represented in Figure 6.

6 . Complexity and Experimental Results

Our method was implemented in CommonLisp in a Sun-Spark environment. We have first tested on a large set of problem instances involving 10 robots and about 50 containers, with variations upon the amount of imprecision, and thus upon the threshold T.

Theoretical study of the complexity shows that the allocation process runtime is stricly bounded, and does only depend upon the amount of imprecision and the number of robots. Experimental results give account of running times between 1 to 10 seconds for the allocation steps, and in the order of a millisecond for the temporal propagations during execution, which has to be compared to durations of actions in the order of a minute at the less. So it appears to be negligible, as far as a sufficient time-lag is maintained between allocation and execution, especially thanks to the tuning of dimension T.

Afterwards, we have experimented with more robots and containers, up to 50 robots. It has been observed that the initial off-line process of expanding the graph, with initial propagations became really costly: in the order of a minute as far as the number of containers exceeds 80, and about 10 minutes for 200 containers. The use of dynamic memory to store this graph structure became also quickly overwhelming. But the global in-line allocation/execution process behaves with nearly the same experimental complexity, in the order of a few seconds, thus paying off for the initial preprocessing.

7 . Conclusion

The approach presented throughout this paper describes techniques for allocating predefined missions to a large team of robots. The imprecise nature of temporal information led us to adopt an interleaved allocation/execution process, which gave birth to the necessity of finding highly efficient temporal management techniques. This was made possible thanks to an application-dependent graph decomposition technique, allowing to take into account complex temporal constraints.

We finally built a robust complete system, with the quality of the solution produced depending upon the amount of imprecision characterising the instance of the application. However, this technique is incomplete in the sense that in complex situations, it may not find a solution when there is one. Nevertheless, thanks to our temporal propagation technique, failing to reach a solution can be discovered early in the global process, and thus constraint relaxation techniques (like for example requesting a delayed departure of a boat) can be activated in advance. Moreover, thanks to our modular architecture, we can improve our heuristic functions in order to be able to take into account those more complex situations.

These encouraging results made it possible to consider the integration of this process into a real application, mixing the high-level global allocation system together with the low-level distributed route planning and execution architecture (such as in [10] for example).

References

1. R. Alami, F. Robert, F.F. Ingrand, S. Suzuki - *Multi-Robot Cooperation through Incremental Plan-Merging*. IEEE Int. Conf. on Robotics and Automation, Nagoya (Japan), 1995.
2. P. M.Berry, B. Y.Choueiry & L. Friha - *Multi-Agent Architecture for a Dstributed Approach to Resource Allocation using Temporal Abstractions*. Technical Report, EPFL, n. TR-92/18, 1992.
3. M. Boddy - *Temporal Reasonning for Planning and Scheduling*. SIGART Bulletin, 4(3), 1993.
4. A. Collinot & C. Le Pape - *Controlling Constraint Propagation*. Proc. 10th IJCAI, Milan (Italy) 1987.
5. A. Collinot, C. LePape, G. Pinoteau - *SONIA: A Knowledge-Based Approach to Industrial Job-Shop Scheduling*. International Journal for Artificial Intelligence in Engineering, 3(2), 1988.
6. T. Dean - *Intractability and Time-Dependent Planning. Proc. «Reasonning About Actions and Plans»*, 1986.
7. J. Erschler, P. Lopez & C. Thuriot - *Temporal Reasoning under Resource Constraints and Scheduling Problems*. Revue d'Intelligence Artificielle, 5(3), 1991 [in french].
8. S. French - *Sequencing and Scheduling: an Introduction to the Mathematics of Job-Shop*. Whiley, 1982.
9. M. Ghallab & T. Vidal - *Focusing on a Sub-Graph for Managing Efficiently Numerical Temporal Constraints*. Technical Report, LAAS-CNRS, 1994.
10. C. Le Pape - *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling*. Proc. IEEE Robotics and Automation, 1990.
11. C. Le Pape - *Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems*. In «Intelligent Systems Engineering», 1994.