ral networks that may be used in any system relying upon such networks and giving account of some particular structure in which a «strict decomposition» may appear.

7. References

[Boddy,93] M. Boddy - *Temporal Reasoning for Planning and Scheduling*. SIGART Bulletin, 4(3), 1993.

[Collinot,87] A. Collinot & C. Le Pape - *Controlling Constraint Propagation*. Proc. 10th IJCAI, Milan (It) 1987.

[Collinot,88] 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.

[Dean,86] T. Dean - *Intractability and Time-Dependent Planning*. "Reasoning About Actions and Plans", 1986.

[Dechter,89] R. Dechter, I.Meiri, J.Pearl - *Temporal Constraint Networks*. Technical Report, Oct 1989.

[Erschler,91] J. Erschler, P. Lopez & C. Thuriot - *Raisonnement Temporel sous Contrainte de Ressource et Problemes d'Ordonnancement*. Revue d'Intelligence Artificielle, 5(3), 1991 [in french].

[French,82] S. French - *Sequencing & Scheduling: an Introduction to the Mathematics of Job-Shop*. Whiley, 1982.

[Ghallab,89] M. Ghallab & A. Mounir-Alaoui - *Managing Efficiently Temporal Relations Through Indexed Spanning Tree*. Proc. 11th IJCAI, 1989.

[Ghallab,94] M. Ghallab & T. Vidal - *Focusing on a Sub-Graph for Managing Efficiently Numerical Temporal Constraints*. LAAS Report 94303, Jan. 1994.

[LePape,90] C. Le Pape - *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling*. Proc. IEEE Robotics and Automation, 1990.

[LePape,94] C. Le Pape - *Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems*. to appear in «Intelligent Systems Engineering», 1994.

[Vidal,94] T. Vidal, M.Ghallab & R.Alami - *Dynamical Allocation of Predefined Tasks to a Large Team of Robots*. LAAS Report n 94303, Aug 1994.

action brings some real duration value that is propagated to the following expected actions, thanks to an arc-consistency algorithm that runs efficiently 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.

One can notice that in fact allocation and execution processes work on the same graph, but not exactly on the same sub-set of time-points: the sliding interleaving technique leads to a certain time-lag between both.

If we have a lot of imprecision, we cannot require a tight discrimination criteria between two robots[i], because the execution would let this imprecision unresolved, and the process would stop. So, we need to adapt our requirement of quality to the level of imprecision. We should tune it, with statistical or learning methods for example, such that each robot is given at each time an horizon of one up to three tasks, thus maintaining a sufficient time-lag between allocation and execution.

6 . Experimental Results and Conclusion

Our method was implemented in CommonLisp in a SunSparc environment. The formal details about the tests can be found in [Vidal,94]. We can summarize our results as follows:

1. With less imprecise data, we get better near-optimal solutions.
2. The temporal propagation processes appear to be negligible compared to the global allocation process, which complexity itself appears to be strictly bounded and runs in the order of the second[ii].

3. For big databases, the initial off-line process of expanding the graph, with initial propagations became really costly. But the global in-line allocation/execution process behaves with nearly the same experimental complexity, thus paying off for the initial preprocessing.

Thus, we assume a robust interleaving process, with the quality of the solution produced depending upon the level of imprecision characterising the instance of the application.

* * *

The application presented throughout this paper describes techniques for allocating predefined identical tasks to a large team of identical robots. This is done at a high-level of abstraction, with centralised techniques, in order to give to each robot a global description of the tasks it has to perform. Path planning and trajectory control capabilities are managed in a distributed way in the ESPRIT project MARTHA.

The need for near-optimal solutions, considering the imprecision of temporal information, led us to adopt an allocation/execution interleaving process, which gave birth to the necessity of finding highly efficient temporal management techniques. This was made possible thanks to

- the separation between heuristic choices and temporal management.
- the use of the same graph, with distinct propagation algorithms, for managing allocation as well as execution steps,
- and essentially, the proof of some important property of the path-consistency mechanism made it possible to restrict it to local propagations, keeping the global completeness of the propagation process with respect to the application needs. This is a general property of tempo-

---

i. formally a short overlapping of the two intervals corresponding to the possible values of the availability times of the two robots

---

ii. which has to be compared to durations of actions in the order of 10 to 15 minutes at the less.

Thus we have to run the two decision steps that are usually addressed in search techniques especially when backtrack-free search is required, and that will be made through heuristic functions that are briefly introduced here (see [Vidal,94] for more details):

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

   We will look for the "most critical" container (in a temporal sense).

   This search runs in linear time in the number of reachable containers, which are generally few, as far as the containers are «highly stacked» in the conveyors.

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

   We will look for the robot that is likely to arrive first at the unloading area.

   This search runs in linear time in the number of robots.

* * *

Once the choices have been made, temporal precedence constraints will be added

• between the last availability time-point of the robot and the beginning of its new task,
• and ordering constraints for the PICKUP and PUTDOWN operations.

This constraints adding process goes with temporal propagations as it is depicted in section 3, leading to updating the graph, and then to new values on the availability date of the robot and the temporal windows on each unloading/loading operation.

In [Vidal,94], we detail a least-commitment approach, mixed with some heuristic choice when needed, to obtain a backtrack-free near-optimal ordering of the unloading/loading operations.

To summarize, the global allocation process runs through the following steps:

1. Heuristic choice of the most critical container: CONT.
2. Heuristic choice of the best robot to convey it: ROB.
3. Strict ordering of the last PUTDOWN made by ROB, with associated temporal propagations.
4. Temporal propagation in the sub-graph of the currently allocated task.
5. Strict ordering of the PICKUP of the currently allocated task, with associated temporal propagations.

One can easily denote the separation between heuristic decisions and temporal management, which allows flexibility of the global system: it is always possible to improve the heuristic functions, without challenging the overall system.

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 tasks, because of lack of precision in the numerical constraints given in input. The more tasks we give to a robot, the more imprecision we get in its new availability time, because of imprecision growing within successive propagation processes. We reach a point when it is no more possible to choose between two robots in a sufficiently deterministic way: 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 provides more precise values: each executed

turn propagated in that neighbour cluster. The process goes on that way until no constraint is modified. Hence we will not compute the complete graph, but we can easily prove that every constraints included in a cluster will be minimal at the end of the process. This intuitive behaviour is summarized through the two following properties, that can be easily proved:

1. The task-graphs defined above represent a strict decomposition of the overall graph of the application.
2. The clusterised propagation is complete: it always detects a global inconsistency if there is one, and moreover provides minimal constraints (thus new precedence constraints as well) within each cluster.

The only constraints requested by the allocation process (see above) appear, from the definition of the task-graph, to be confined to those task-graphs. Thus we can say that our clusterised propagation process behaves in the same way as a global propagation would do, regarding the global 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. As we only allocate within a short-term horizon, the total number of clusters in the graph corresponding to the tasks being allocated is strictly bounded, thus k is strictly bounded, which leads to nearly constant time propagation process.

To end with this section, let us look at the global temporal management process during the allocation process. At the beginning, the global graph is expanded, with all the task-graphs in parallel. Initial propagation in each sub-graph (not yet connected one to another) is also made.

Each time a robot is allocated to a mission, the precedence constraints corresponding to the interactions with other clusters are added (as in our graph example), launching clusterised propagation steps. Thus, the global graph evolves from a highly parallel one to a more sequential one, where, as usually in scheduling problems, tasks for a robot are incrementally ordered, and constraints between those robot sequences of tasks appear because of the ordering of the unloading/loading operations.

4 . The Allocation Decision-Making Loop

Just remember that in our interleaving approach, we have to dynamically allocate one robot to predefined tasks of handling containers, until some threshold in time is reached, that will require to wait for results of the execution of the allocated task (see part 5).

So we will define a loop whose incremental steps are allocation of one robot to one container, with the aim of not backtracking on these decisions. At each of these steps, we are given:

- (a) the current partially ordered overall graph,
- (b) the current position and «availability time» of each robot (i.e. the resources availability),
- (c) the current stacks of containers remaining on each conveyor, and the loading area for each container (i.e. the set of tasks requiring sequencing and resource allocation).

We want to get some near-optimal (or «good quality») solution according to the criteria which appears to be preeminent in our application: minimising the earliest end time of the overall plan (as in [LePape,90]). In other words, we wish to process all the unloading/loading tasks 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 leaving.

Thus, the global graph appears to be composed of task-graphs like in the first figure of last page, some of them connected one to another by precedence constraints.

Let us define more precisely those task-graphs: each task is associated to a cluster of 14 time-points containing:

• the origin of time 0 (reference frame of the dates),
• the end time-point of the last task assigned to the same robot, if there is one.
• the initial and final points of each action in the task (8 time-points),
• the current temporal windows for PICKUP and PUTDOWN actions (4 time-points).

Those intersecting clusters are represented in the second figure in last page, by means of circles, not including, for clarity concerns, the time-point 0. For the same reason, the precedence constraints between 0 and some conveyor availability beginning time-points are not represented. In fact, the only temporal constraints that are useful for the global mission allocation process, as we will see in next part, are:

• the date of availability of a robot, i.e. the date of the end of the last task allocated.
• the temporal windows for the PICKUP and PUTDOWN operations.

This means that the only minimal constraints that are useful are constraints between time-points belonging to the same cluster. In other words, we will never need to know the temporal distance between the beginning of PICK-UP(Rob1,Cont2,Boat) and the end of GOTO(Rob4,Park1,Train), for example. Thus we only need to get minimal constraints within those clusters to guide the allocation decisions.

Let us now briefly present the properties that are at the heart of our process:

« **Definition 1** »
• A **strict minimal precedence** (i,j) is a precedence constraint that cannot be entailed by another (by transitivity).
• A «**strict decomposition**» D of a graph G is a set of clusters $(C_u)_{u=1,...c}$, corresponding to overlapping sub-sets of time-points $\{i_{u_1}, ..., i_{u_m}\}$, such that:

   $\forall i \in G, \exists\, C_u$ such that $i \in C_u$ (complete partition upon time-points),

   $\forall i \in G, \forall j \in G$, such that (i,j) is a strict minimal precedence, then necessarily $\exists$ at least one u such that $i \in C_u$, and $i \in C_u$ (complete partition upon constraints).

• The «**neighbour clusters**» of a cluster C_u are $C_{v_1}, ..., C_{v_m}$ such that for each $C_{v_j}, \exists$ a time-point i_{v_j} such that $i_{v_j} \in C_{v_j}$ and $i_{v_j} \in C_u$ at the same time.

The «clusterised» propagation process will then be as follows:

• If D is a strict decomposition of G, if (i,j) is a temporal constraint being modified, then the propagation process will be the following recursive process:

   execute a propagation step within C_u and within each of its neighbour clusters $C_{v_1}, ..., C_{v_m}$, and then repeat the same process for each modified cluster, until there is no cluster modified.

Let us illustrate those definitions through the graph in the second figure of last page. The arrows represent strict precedences of the graph. We can see that all of them are included in at least one cluster. Then, a modification of such a constraint will be surely propagated to all the other constraints of the cluster, that will be updated. Some modified constraints are included in other clusters as well: those are the precedence constraints between tasks, or the ordering constraints between PICKUP/PUTDOWN actions (in dotted lines in the figure). They are in

all the minimal constraints (i.e. where only values globally consistent with the other constraints are kept), and at the same time gives account of new precedence constraints. As our application involves large number of robots and containers, n is quickly in the order of a few thousands time-points (about ten times as many as the number of containers). Hence a cubic complexity will end in unbearable time consuming algorithms.

3 . The Temporal Graph Decomposition Scheme

We can take some advantage of the particular structure of the temporal network in our application. The global decision-making process consists of:

• allocating each robot to successive tasks, which leads to a strict ordering of tasks for each robot,
• and adding constraints on PUTDOWN and PICKUP operations (following the constraints on crane actions), thus inducing precedence constraints between tasks.
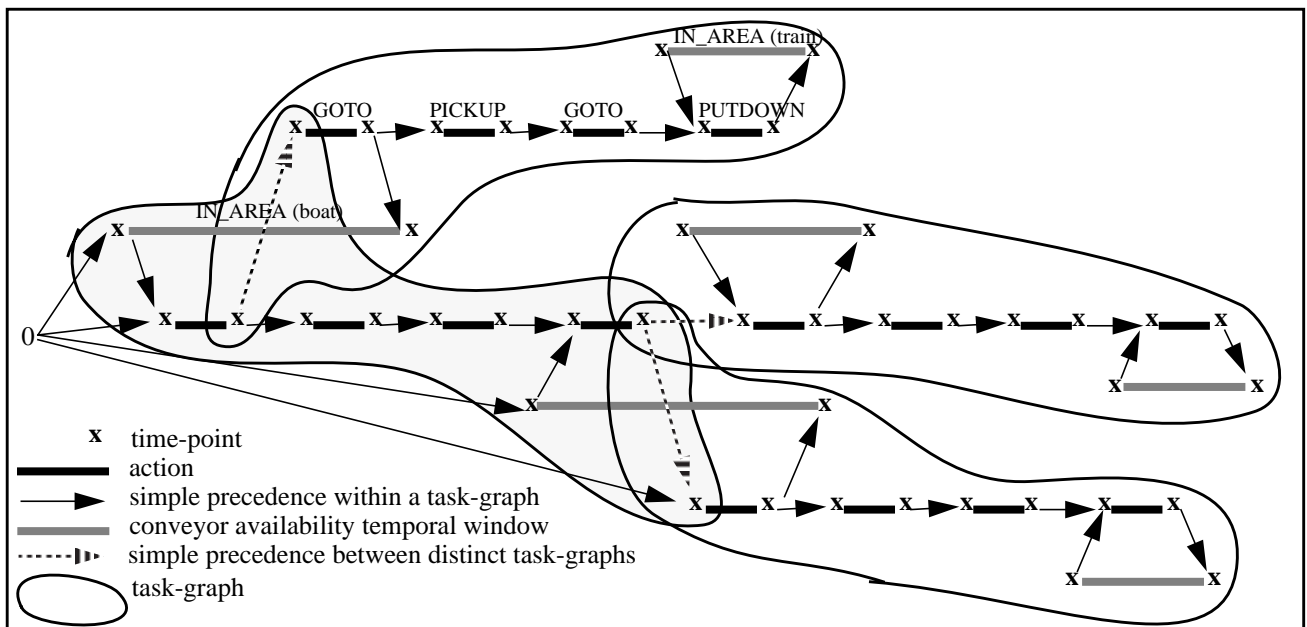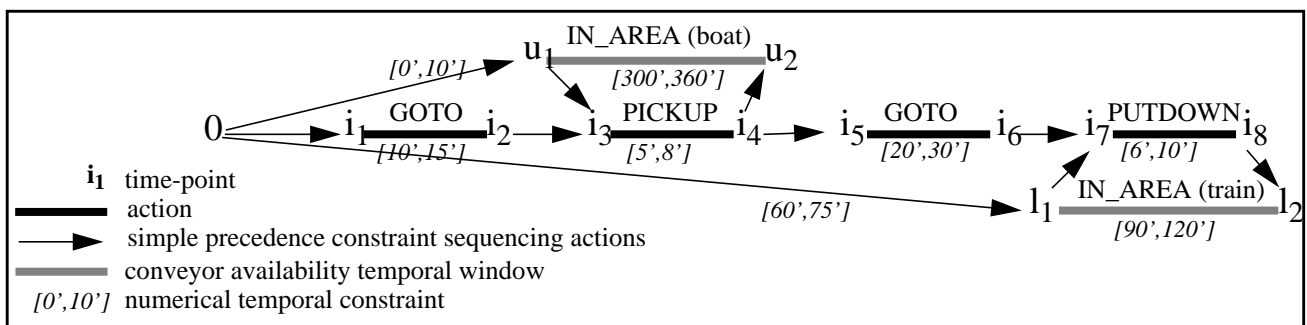
Our proposal is to decompose the global temporal graph into sub-parts (or clusters) in which propagation can be confined. Our decomposition scheme relies upon the application-dependent structure of the graph, whereas for example [Dean,86] proposes a temporal decomposition based upon complex problem characterisation techniques. We prove that our method meets some important property that keeps its algorithmic completeness (i.e. it always detects an inconsistency when there is one); it also gives back the minimal constraints [Dechter,89], when considering only the constraints that are required by the decision-making process. We then achieve a highly reliable temporal constraints manager whose propagation runs in nearly constant time, keeping at the same time the total expressiveness of the [Dechter,89] temporal constraint networks.

One important thing to notice is that we take advantage of a separate temporal constraint management system, which can be compared to the [Erschler,91] system and also to [Boddy,93] system relying on the Dean's TMM. Another important feature is that our method is incremental, which means that ordering constraints are added one by one, updates and consistency checking being made at each step.

2 . Application Context and Basic Representation Issues

The global mission in our application domain is the following: a large team of robots have to load/unload ships in a harbour (ESPRIT project MARTHA). Containers (a few hundreds per mission) arrive in boats or trains (we will use the generic term of «conveyor»). They are to be brought to some other place: stockage areas, or conveyor areas. Those tasks are to be carried out by a crew of robots (about 50), all identical, through predefined routes linking the different areas in the harbour. A mission allocation system has to decide which robot is going to take care of which container. The generic task performed by a robot is a sequence of the four actions:

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

Each of these actions have an associated duration, and the PICKUP and PUTDOWN actions have to be made during the availability temporal window of the corresponding conveyor.

The containers are arranged inside a conveyor by stacks, which defines an initial partial order on unloading operations. But there is just one crane per area, which means that PICKUP and PUTDOWN operations have to be strictly ordered in the final plan for each area. Moreover, cranes are not explicitly represented but implicitly managed throughout the temporal ordering of PICKUP/PUTDOWN operations.

* * *

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, see [Ghallab,89]), and also by imprecise numerical constraints (durations and dates) given as possible durations labelling the precedence edges (see [Dechter,89] and [Ghallab,94]). This leads to represent the above general task with the conveyors availability temporal windows as it is shown in the first figure of next page.

As far as numerical constraints are concerned, a path-consistency propagation algorithm, running in $O(n^3)$, n being the total number of time-points in the overall graph, is required. It allows to check for global consistency, making explicit

Efficient Temporal Management through an Application-Dependent Graph Decomposition

Thierry Vidal

thierry@laas.fr

Malik Ghallab

malik@laas.fr

LAAS-CNRS - 7, avenue du Colonel-Roche, 31077 Toulouse, France

Abstract

In the MARTHA project, a large number of robots in a harbour are given the global task of transporting standardized containers from one area to another (ships, trains, stocking areas). The global decision-making process consisting 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 (expected arrival and leaving times of ships and trains, durations of actions) 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 large applications as our, the complexity of temporal management is a crucial issue. We present here a technique of graph decomposition, leading to nearly-constant time temporal propagation, without any loss of information. We finally relies on a complete, highly expressive and efficient temporal reasoner used by our global decision-making loop.

1 . Introduction and Related Work

When addressing scheduling and resource allocation problems in the context of real applications involving dynamical uncertain worlds, one has to manage temporal constraints as well as resource constraints, trying to get a robust schedule of good quality as fast as possible.

In our application context, we have to allocate predefined tasks to a number of identical robots. It is in fact what [LePape,94] calls a joint problem, mixing constraint-based scheduling and resource allocation. We are mainly concerned with the imprecision of temporal constraints on expected events and goals. To decide which robot to allocate to a task, we use heuristic functions approximating a given optimality criteria along a greedy search approach, thus leading to near-optimal solutions. Here, the global objectives that have to be met are temporal ones (earliest end time of the overall plan), hence the choices are guided by the temporal constraints. As those are imprecise, short term predictions and objectives are usually quite reliable whereas long term ones are not. This lead us to adopt a dynamic approach (as it is defined in [French,82]), interleaving task allocation and execution, like in [Collinot,88], or in [Dean,86].

The efficiency of the allocation process is a crucial issue. Like in [Erschler,91] or in [Collinot,87], 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 [Erschler,91] as most authors use an $O(n^3)$ complexity algorithm that is too expensive in our application, whereas [Collinot,87] uses focusing techniques that, although enhancing the propagation efficiency, do not achieve its completeness any longer, and then leads to «deviations» in the decision process.