A Coordination Mechanism to Solve Common Resource Contention in Multi Agent Space Systems

Andrea Brambilla and Michèle Lavagna
Politecnico di Milano, Dipartimento di Ingegneria Aerospaziale
brambilla@aero.polimi.it lavagna@aero.polimi.it

Abstract

This paper presents an approach for coordinating the usage of common resources in a team of intelligent space systems. This problem typically arises in multi agent scheduling where each task is controlled by a specific unit and requires a certain amount of a limited capacity common resource. Just to state an example we can report the scenario of a rovers team contending for a common orbiter antenna as unique data relay to Earth. The coordination is fulfilled by decoupling the global interdependencies with proper local constraints. The advantage consists in limiting the communications at execution time and in reducing the ripple effects of distributed backtracking.

1. Introduction

An increasing interest is posted on autonomous control of remote space systems (e.g. rovers) working concurrently to accomplish mission goals. Each system can be modelled as an intelligent agent provided with coordination capability. Within this context a relevant technology is the distributed scheduling: every agent has the responsibility to allocate on a temporal horizon the set of activities that are under its own control. The allocation must satisfy a set of operative constraints to be feasible. In particular, the presence of global constraints (like the demand for a common limited capacity resource) asks for the establishment of communications to converge at a coordinated schedule. An example representative of this class of constraints is an orbiter antenna accommodating only one link in a rover team at once.

This paper presents an automatic coordination mechanism to solve the contention for a common resource in distributed scheduling. The general idea of our approach is the following. Iteratively the agents commit a sequencing constraint between two bottleneck activities in order to reduce the shared resource contention level. The committed constraint (when involving activities controlled by different agents) represents an explicit interdependency that can require significant communication costs to be satisfied both during the scheduling process and runtime. Our mechanism decouples this interdependence which is made redundant by posting local constraints. The temporal decoupling methodology is reported in [1]. Among the spectrum of all possible decoupling ways is committed the choice which most retains temporal flexibility. One benefit is the reduction of the undoing of scheduling decisions, thereby avoiding the ripple effects of distributed backtracking. Another benefit is the communication saving as the decoupling constraints are local conditions for the agent that implicitly satisfy coordination on common resource usage.

Performances on a suite of problems run on a Linux workstation network is also reported.

2. Problem definition and assumptions

We assume a team of agents working in a common environment. Each one is responsible for controlling (executing) a portion of a global plan. None has a global view of the scheduling problem but every agent has knowledge of:

1. its own controllable activities
2. the interdependencies with uncontrollable activities (i.e. controlled by another unit)
3. uncontrollable variables subject to interdependencies with the agent itself (e.g. a task of a different agent demanding for a common resource).

We consider unary and cumulative resources. In scheduling literature a cumulative resource is used over some time interval: a certain quantity of resource is consumed at the start time of the activity and the
same quantity is released at its end time. A unary resource is a cumulative resource with unit capacity. It imposes that all the activities requiring the same unary resource are totally ordered. This is typically the case of an antenna that can process only one communication link at once. A survey on resource constraint handling in scheduling problems is [2].

We also assume that each resource is either commonly shared by the team (e.g. a single antenna) or locally available to the agent (e.g. the on-board power).

The only kind of interdependency among the agents is in terms of shared resource contention. The scope of the problem is to obtain for each agent a feasible and temporally flexible schedule of the controlled activities. Such a schedule has to satisfy also the coordination requirements committed by the team for solving the contention on common resources usage. As contention we mean the possibility that for some consistent time assignment to the activities there is at least one time at which the total amount of resource allocated is outside the availability bounds.

3. Overview of approach

A classical scheduling approach works by binding time values to the starting time of each activity (ending time is dependent assuming a fixed duration). A feasible schedule is reached by either extending a partial consistent solution or repairing an inconsistent global assignment. The solution found is a temporally fixed activity schedule typically optimizing an objective function like the make-span.

An alternative approach is described in [3] and has been also adopted for this work. It does not bind exact values to variables but adds sequencing constraints among activities in contention for the same resource. The scope is to post enough constraints to ensure that at any point in time the requested capacity does not exceed the available capacity. According with this approach the search space is not represented by the temporal domains of the variables but is the set of all possible orderings among activities in contention for a same resource. The final result is not a fixed time schedule but a temporal constraints network that satisfies all the resource capacity limitations. The network represents a flexible behaviour able to better absorb the execution uncertainties and to better accommodate new scheduling requests like the addition of a new task. An execution controller algorithm working on temporal constraint networks is reported in [4].

Within a multi agent context we have to consider resources shared by the team and we need to distinguish two cases in posting an ordering constraints between activities in contention. A sequencing constraint can be posted either between activities controlled by the same agent or among activities owned by different units. This latter case is an interdependency added between different agents and hence a coordination method must be established.

We adopted the temporal decoupling method to coordinate resource interdependencies. Roughly speaking, the idea is to decouple a sequencing constraint whenever it is posted between activities in contention and controlled by different agents. The decoupling of an ordering is obtained by making it redundant posting proper local constraints. In this way, at execution time, the agent needs only to keep coherence of local impositions to be automatically coherent with the team. There exists a spectrum of possible ways to decouple an ordering relation and a drawback of this approach is the loss of schedule flexibility as new further constraints are added for decoupling. We will present a new heuristic iteratively committing the ordering constraint that mitigates the loss of temporal flexibility. As outcome, each agent is provided with a temporal constraint network which satisfies the limitations both on local and shared resources with the further property to be independently managed at execution time. This property is a direct consequence of the temporal decoupling approach.

4. The coordinated scheduling

In this section we will first describe the underlying representation for the temporal constraints and resource consumptions of the activities, introducing the consistency checking methods adopted for flexible schedules. We will further describe the scheduling algorithm pointing out the coordination mechanism.

4.1. The STN based representation augmented with resource consumption

Each agent framework relies on temporally flexible schedules and uses the Simple Temporal Network (STN) representation as temporal reasoning engine [5]. An STN represents temporal constraints as a graph $G \langle N, E \rangle$. The nodes in $N$ are the set of time points (i.e. the starting and ending times of activities) while weighted edges in $E$ are temporal distances between pairs of nodes in $N$. A special node $z$ represents the origin of the temporal horizon. It grounds the network and has the value set to zero. A wide array of
requirements (e.g. release time, deadline, orderings, durations) are uniformly represented within this model as graph weighted edges. When new temporal decisions (e.g. new activity orderings) are committed the network propagates the introduced constraints and maintains lower and upper bounds on all time points in the network. This is accomplished efficiently via the use of a standard all-pairs shortest path algorithm. As bounds are updated, a consistency check is made (verifying the presence of negative cycles into the graph), and the absence of any such cycle ensures the continued temporal feasibility of the network (and hence the schedule). Otherwise a conflict has been detected, and some amount of constraint retraction is necessary to restore feasibility.

Dealing with flexible schedules has the difficulty of accurately estimating the amount of resources that a flexible plan may need across all of its possible executions. We fulfill this need by computing the maximum resource envelope. This means that for any possible time value we will compute the maximum possible consumption among all possible schedules. We adopted the method described in [6] which is a polynomial time algorithm for maximum resource envelope computation.

The formal model instrumental for this purpose is the cR-STN (piecewise constant Resource-Simple Temporal Network). Unlike regular STNs, each event (node) has an associated allocation variable (e.g., $r_{31}$ for event $e_{3s}$ in fig.1) with real domain representing the amount of resource allocated when the event occurs. An event with negative allocation is a consumer, while an event with positive allocation is a producer for a single resource.

Figure 1 shows an example of the cR-STN based representation. The network has two time variables per activity, a start event and an end event (e.g., $e_1s$ and $e_1e$ for activity $A_1$), a non-negative flexible activity duration link (e.g., $[2, 5]$ for activity $A_1$), and flexible separation links between events (e.g., $[0, 4]$ from $e_{3e}$ to $e_{4s}$). The time origin $T_s$ corresponds to time 0. Links and event nodes represent a STN.

We have introduced so far the two adopted consistency checking methods for flexible schedules:

1) negative cycles on STN for temporal consistency.

2) resource maximum envelope for threshold crossing control.

![Figure 1. A STN with resource allocation. This example is taken from [4]](image)

4.2. The search algorithm

Several scheduling systems work by binding values to the problem variables (i.e. the starting and ending times of activities). A feasible schedule is reached by either extending a partial consistent solution or repairing a total not consistent assignment.

The alternative search process that has been here adopted operates by adding sequencing constraints among activities in contention for the same resource. The goal is to post enough constraints to ensure that at any point in time, the requested capacity does not exceed the available capacity.

Within a multi agent context, posting a sequencing constraint between two activities presents further complications. In fact, the activities can be controlled by different agents and a sequencing constraint addition means a new interdependency to be handled and coordinated. An earlier work related to this topic is [7].

The scheduling process is shown in fig.2 as pseudocode layout and is described in the reminder of the paper.

```plaintext
1: R /* Agent cR-STN */
2: T /* Search tree data structure */
3: Envelope /* envelope of the resource r. It is a list of three items < L, P > where L is the envelope level at time t and P is the set of unranked pending events at t contributing to L */
4: R_sh /* the set of shared resources */
5: R_loc /* the set of local resources */
6: cap(r) /* available capacity of the resource r */
7: singleResCont /* list of pairs < x, y > x and y are two unranked pending events in contention. The list is generated for a given resource */
8: allResCont /* list integrating all the singleResCont */
9: globalCons = false;
10: localCons = false;
11: while( ! (globalCons & ! localCons ) )
12: {
13:    singleResCont is set to empty;
```
for each \( r \in R_{ebu} \)

16: \{
17: envelope = Resource_Envelope( R, r );
18: singleResCont = Cap_Crossing( envelope, cap(r) );
19: allResCont = Append( allResCont, singleResCont );
20: } 

21: ...
22: if ( ls_Empty( allResCont ) ) globallyCons = true;
23: else globallyCons = false;
24: ...
25: if ( globallyCons ) locallyCons = Local_Scheduling( R, R_{ebu}); /* Local scheduling of the decoupled partial solution */
26: else T = Branching( T, allResCont ); /* The search tree is extended in depth with a new branching level */
27: ...
28: 
29: p = Commitment( T ); /* p is a sequencing constraint between two events. The corresponding node in T is deleted */
30: isCons = false; /* flag controlling the consistency check for p */
31: while ( ! isCons )
32: 
33: < isCons, R > = Coordination( R, p ); /* If propagating p brings to an inconsistency isCons is set to false and R is returned with no modifications */
34: if ( ! isCons ) p = Backtracking( T ); /* The corresponding node in T is deleted */
35: 
36: 
37: }

**Figure 2. Pseudocode of the algorithm**

The contention on shared resources is first addressed. For each resource \( r \) the envelope is computed (line 17 fig.2). The available capacity consistency control is then performed (line 18 fig.2). For each time \( t \) having a not consistent envelope level for \( r \) the corresponding set \( U(t,r) \) is computed. It is the set of unranked events having a temporal domain crossing \( t \). The all possible pairs \((x, y)\) where \( x, y \in U(t_r, r) \) are generated and returned.

At line 19 of fig.2 the list \( U \) aggregates the pairs obtained by the sets \( U(t_r, r) \forall t_r \forall r \). Every pair is considered only once into the list \( U \) even if repeated in more sets \( U(t_r, r) \).

The posting of sequencing constraints among unranked pending events levels down the envelope value. Each event is classified as producer or consumer for a single resource (see par.4.1). We extend this classification considering all the resources: an event is globally producer \( globProd(e) \) if is a producer for the greater part of \( R_{glob} \). The definition of globally consumer \( globCons(e) \) is straightforward. This definition is instrumental for defining the branching scheme.

For each pair \((x,y)\in U\) we branch on the sequencing constraints \( t(x) \leq t(y) \) or \( t(x) > t(y) \) (line 26 fig.2) as summarized in tab.1.

\( t(x) \) denotes the date variable for the event \( x \).

**Table 1. Branching scheme**

<table>
<thead>
<tr>
<th>( globCons(x) )</th>
<th>( globProd(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( if )</td>
<td>( if )</td>
</tr>
<tr>
<td>( Flex(t(x)&gt;t(y)) )</td>
<td>( Flex(t(x)&gt;t(y)) )</td>
</tr>
<tr>
<td>( try ( t(x)&lt;t(y) )</td>
<td>( try ( t(x)&lt;t(y) )</td>
</tr>
<tr>
<td>( then ( t(x)&gt;t(y) )</td>
<td>( then ( t(x)&gt;t(y) )</td>
</tr>
<tr>
<td>( else try ( t(x)&gt;t(y) )</td>
<td>( else try ( t(x)&gt;t(y) )</td>
</tr>
<tr>
<td>( then t(x)&lt;t(y) )</td>
<td>( then t(x)&lt;t(y) )</td>
</tr>
</tbody>
</table>

The metric \( Flex \) of a sequencing constraint is an estimation of the retained temporal flexibility after posting the constraint. Let \( x \) and \( y \) be two unranked events with respective lower and upper bound for time value: \( t_{min}(x), t_{max}(x), t_{min}(y), t_{max}(y) \). The value for \( Flex \) in posting the constraint \( t(y) \leq t(x) \) (i.e. event \( y \) before \( x \) can be estimated as the ratio of the area of the rectangle \( t_{min}(x), t_{max}(x), t_{min}(y), t_{max}(y) \) that is preserved by the constraint as illustrated in fig.3. An analogous estimation is made in [8].

**Figure 3. The grey area is preserved after posting the constraint \( t(y) \leq t(x) \)**

Fig.3 shows the case where \( x \) and \( y \) are both controlled by the same agent and so decoupling is not considered. The fig.5 shows the case of different controllers for \( x \) and \( y \) (the preserved area has to take
into account the addition of local decoupling constraints).

In line 33 of fig.2 the committed constraint is propagated into the local STN. Two possibilities arise at this step: a sequencing constraint can be added between activities controlled either by the same unit or by different entities. In the latter situation, the new constraint is decoupled introducing proper local constraints. According with this coordination strategy each agent reaches a partial solution which satisfies the shared resource contentions and is also decoupled by the partial solutions of the other agents.

The decoupled partial solutions can therefore be processed by a second flow devoted to address the local resource contentions (line 25 of fig.2). This second stage can be locally performed by each agent without communications as only local constraints are present.

4.3. Coordination through decoupling

This section focuses the coordination issue.

The basic idea is decoupling the interdependencies added to solve shared resource contention. They are in the form \( t(y) \leq t(x) \) (i.e. event \( y \) before \( x \)) and represent a sequencing relation between \( x \) and \( y \). This constraint is made redundant through the addition of proper local constraints. Here is reported an example taken from [9] just to give an understanding of the approach.

Two time-points, \( x \) and \( y \), are subject to the following constraints: \( 0 \leq y \leq x \leq 10 \). The corresponding STN representation is shown in the left-hand side of fig.4.

Figure 4. Temporal decoupling example

The goal is to distribute control over the network to two agents, \( A \) and \( B \), such that \( A \) controls the execution of \( x \), and \( B \) controls the execution of \( y \). As things currently stand, \( A \) and \( B \) cannot be given independent and complete control over their respective subnetworks; as soon as they execute their time-points independently, they may violate the constraint that \( y \) be executed before \( x \). The right-hand side of fig.4 shows a possible result of the temporal decoupling on the STN at the left-hand side of the figure. Notice that the edge from \( x \) to \( y \) (which represents an inter-subnetwork constraint) has been made redundant by the new path \((x; z; y)\). So the decoupling constraints are:

\[
\begin{align*}
 y - z & \leq 4 \text{ for agent B} \\
 z - x & \leq -4 \text{ for agent A}
\end{align*}
\]

For an exhaustive description of the temporal decoupling see [1].

Among the all possible ways to decouple an ordering constraint posted between two variables \( x \) and \( y \) we consider the one maximizing the value for the metric \( \text{Flex} \).

Figure 5. The grey area is preserved after decoupling the constraint \( t(y) \leq t(x) \)

The value of \( \delta \) can be computed through a simple analytic optimisation of the function:

\[
f(\delta) = [t_{\text{max}}(x) - \delta - t_{\text{min}}(x)][t_{\text{min}}(x) + \delta - t_{\text{min}}(y)]
\]

which is the grey area in the size of \( \delta \). Posting the derivative to zero \( f'(\delta) = 0 \) we obtain the value:

\[
\delta = \frac{t_{\text{max}}(x) - 2t_{\text{min}}(x) + t_{\text{min}}(y)}{2}
\]

that can be computed in constant time. The decoupling constraints are those reported in solid arrows in fig.6.

Figure 6. Decoupling constraints
The decoupling constraints are propagated in the STN which is in general different from agent to agent so it is not guaranteed an univocal result in terms of consistency check. Every agent broadcasts the its own consistency status following the propagation of the committed constraint. If at least one agent rejects the committed constraint as not consistent, all the team backtracks to the next decision in the search tree (line 34 in fig.2). The agents explores the same search tree while solving the shared resource contention problem. They commit-propagate-verify the consistency of each decision in a synchronous fashion.

4. Experimental results

In this section we compare the performance of the proposed approach across the number of agents considered in the agency. We randomly generated a set of 100 problems. This set of problems has been solved respectively with 1, 2, 3, 5, 6 and 10 agents. Each problem consists of 30 activities all contending the same cumulative shared resource $R$. Each problem has been generated according with the following features:

1. The available capacity for $R$ is 100%
2. The fixed duration for each activity is randomly taken in the range $[1,10]$  
3. The capacity of $R$ depleted by each activity is randomly taken in the range $[1,100]$  
4. The make-span is fixed to 1000 temporal units. 
5. The activities are equally distributed to the agents. (This is guaranteed by considering 30 tasks and the following sizes for the agency: 1, 2, 3, 5, 6, 10). 
6. All the 30 activities are initially constrained to range within the temporal bound $[0,1000]$ and no other temporal dependencies are posted.

The performances are compared according with the two following parameters: the RMS (Root Mean Square rigidity of an STN) and the total committed sequencing constraints needed to reach the solution. For a formal definition of RMS we remind to [1]. This parameter provides a global measure of the temporal rigidity of the STN. The greater is the rigidity the more difficult is integrating new tasks in the schedule or absorbing delays in execution. Tasks subject to flexible constraints can be moved around to accommodate new requests. The rational for choosing the RMS is connected to measure the badness of the final STN.

The second parameter ($Comm$) is the total number of constraints committed during the search process. It provides also an estimation of the total number of communications required to reach the solution. In fact, according with the algorithm previously described, each constraint is synchronously checked by each agent and the corresponding consistency status is broadcasted.

Just to clarify the way this approach works we report here a simple example. Consider three activities: job-1 (duration 9, capacity required 99%), job-2 (duration 7, capacity required 16%), job-3 (duration 2, capacity required 13%) all ranging in $[0,100]$. Job-3 is controlled by Agent2 while job-1 and job-2 by Agent1. Fig.7(a) shows the initial situation.

![Figure 7. Coordination example](image)

The algorithm first commits the ordering of the two jobs controlled by Agent1 without decoupling (fig.7(b)). This decision is the best one for temporal flexibility retaining. At the next step the algorithm
decouples the ordering job-3 before job-2 (fig.7(c)). The final step is decoupling the ordering job-3 before job-1 leaving the two jobs controlled by the same unit ranging in a common temporal interval under a sequencing relation (fig.7(d)).

The decision to decouple job-1 before job-3 would provide more tight domains for the two tasks and so it was not the first choice to explore. We could also reach the same solution skipping the step in fig.7(c) and so reducing the number of the decisions expanded during the search. This is still a weak point of the search strategy we need to address.

In fig.8 we report the mean value for %RMS over the 100 random problems. %RMS is the RMS percentage increment of the global STN after the coordination.

The case labeled as #1 concerns with one agent and represents the centralized approach where no decoupling occurs.

In fig.9 we report the mean value for Comm over the 100 random problems generated. Comm represents the total amount of sequencing constraints committed along the search process. This value is also representative of the total communications as the agent consistency status is broadcasted after the sequencing check. We can notice a decrement of Comm increasing the size of the agency. This can be justified by the fact that more tight temporal bounds are obtained through the search process so can occur that some events in contention can be unranked without posting an explicit ordering relation.

**Figure 8. %RMS with respect to the number of agents**

**Figure 9. Comm with respect to the number of agents**

6. Conclusions and future directions

This paper presented a coordination mechanism to solve common resource contention in multi agent scheduling problems.

The approach is based on temporally decoupling those sequencing constraints posted between activities in contention and controlled by different agents.

As the decoupling affects the temporal flexibility of the schedule an heuristic mitigating such effect has been proposed.

The agents explore the search tree of possible sequencing constraints in a synchronous fashion broadcasting the local consistency status consequent upon each decision node. The communication traffic is so related with the number of sequencing commitments needed to reach the final solution. In order to reduce this feature, future work will investigate the employment of pruning techniques for the set of feasible sequencing alternatives at each stage of the search. A work that couples the sequencing constraint posting with a pruning technique of the space of possible ordering assignments is [10].

Another direction will focus on exploring different choices for the heuristic adopted to fix the decoupling.

From the experimental point of view the approach will be tested on more complex problems, introducing local resources and more than one common resource.

7. References


