Computing the Envelope for Stepwise-Constant Resource Allocations

Nicola Muscettola
NASA Ames Research Center
Moffett Field, California 94035-1000
mus@email.arc.nasa.gov

Abstract. Computing tight resource-level bounds is a fundamental problem in the construction of flexible plans with resource utilization. In this paper we describe an efficient algorithm that builds a resource envelope, the tightest possible such bound. The algorithm is based on transforming the temporal network of resource consuming and producing events into a flow network with nodes equal to the events and edges equal to the necessary predecessor links between events. A staged maximum flow problem on the network is then used to compute the time of occurrence and the height of each step of the resource envelope profile. Each stage has the same computational complexity of solving a maximum flow problem on the entire flow network. This makes this method computationally feasible and promising for use in the inner loop of flexible-time scheduling algorithms.

1 Resource Envelopes

Retaining temporal flexibility in activity plans is important for dealing with execution uncertainty. For example, flexible plans allow explicit reasoning about the temporal uncontrollability of exogenous events [11] and the seamless incorporation of execution countermeasures. Fixed-time schedules (i.e., the assignment of a precise start and end time to all activities) are brittle and it is typically very difficult to exactly follow them during execution. For an example of the effect of fixed-time schedules in an intelligent execution situation, consider the "Skylab strike" [6], when during the Skylab 4 mission, astronauts went on a sit-down strike after 45 days of trying to catch up with the exact demands of a fast paced schedule with no flexibility for them to adjust to the space environment.

A major obstacle to using flexible schedules, however, remains the difficulty of computing the amount of resources needed across all of their possible executions. This problem is particularly difficult for multiple-capacity resources (such as electrical power) that can be both consumed and produced in any amount by concurrent activities. Techniques have been developed [5] [10] for giving conservative estimates of the resource levels needed by a flexible schedule, yielding both an upper bound and a lower bound profile on the resource level over time. In the context of a systematic search method to build flexible plans, resource-level bounds can be used at each search step as follows: a) as a backtracking test, i.e., to determine when the lower/upper bound interval is outside of the range of allowed resource levels at some time and therefore no fixed-time instantiations of the plan is resource-feasible; and b) as a search termination test, i.e., to determine when the lower/upper bound range is inside the range of allowed resource levels at all times and therefore all fixed-time instantiations of the plan are resource-feasible.

Bound tightness is extremely important computationally since a tight bound can save a potentially exponential amount of search (through early backtracking and solution detection) when compared to a looser bound. In this paper, we discuss how to compute the *resource-level envelope*, i.e., the measure of maximum and minimum resource consumption at any time for all fixed-time schedules in the flexible plan. At each time the envelope guarantees that there are

two fixed-time instantiations, one producing the minimum level and the other the maximum. Therefore, the resource-level envelope is the tightest possible resource-level bound for a flexible plan since any tighter bound would exclude the contribution of at least one fixed-time schedule. If the resource-level envelope can be computed efficiently, it could substitute looser bounds that are currently used in the inner core of constraint-posting scheduling algorithms (Laborie 2001) with the potential for great improvements in performance.

To appreciate the difficulty of computing the resource level, we can compare the cases of a fully flexible plan with that of a plan with a single fixed-time instantiation. In the fixed-time case, the envelope degenerates to the resource profile that is used in the inner loop of traditional scheduling algorithms [2][15]. Computing from scratch a resource profile is cheap. It is easy to see that its worst cast time complexity is **O (N lg N)** where **N** is the number of activities in the flexible plan. Consider now a fully flexible plan and assume that we naively wanted to compute the resource-level envelope for a flexible plan by simply enumerating all schedules and taking the maximum/minimum of all resource levels at all times. Since a flexible activity plan has a number of possible instantiated fixed-time schedules that is exponential in the number of events, such naïve method is clearly impractical in most cases.

Note that "resource-level envelope calculation" is *not* equivalent to "scheduling with multiple capacity", an NP-hard problem. For example, note that a reduction of the envelope calculation to scheduling is not straightforward. Consider the interval between minimum and maximum envelope at a given time. It is true that a solution to the scheduling problem can be obtained in polynomial time if the envelope interval is always completely contained within resource availability (in which case all fixed-time instantiations are legal schedules). However, if at least two times the envelope interval is partly outside the availability bound, one can generate examples in which the envelope cannot tell us whether there is a fixed-time schedule that is within availability at all times or whether for all schedules the resource level is outside availability at some time. Discriminating between these two cases still requires search.

This paper presents a polynomial algorithm for the computation of resource-level envelopes based on a novel combination of the theory of shortest-paths in the temporal constraint network for the flexible plan, and the theory of maximum flows for a flow network derived from the temporal and resource constraints. We develop the theory, show that the algorithm is correct, and that its asymptotic complexity is **O(N O(maxflow(N)))**, where **N** is the number of start/end times of the activities in the plan, which is at most twice the number of activities, and **O(maxflow(N))** is the complexity of a maximum flow algorithm applied to an auxiliary flow network with **N** nodes. We believe that this method will be efficient in practice, since experimental analysis [1] show the practical cost of maxflow to be as good as $\mathbf{O}(\mathbf{N}^{1.5})$. However this paper is a theoretical contribution and a definitive answer to its practical complexity will require further experimental work.

In the rest of the paper we introduce some notation and describe the formal model of activity networks with resource consumption. Then we review the literature on resource contention measures and show an example in which the current state of the art is inadequate. The discussion of our algorithm follows. Some informal examples to establish an intuitive understanding of our method are first given. Then we establish the connection between maximum flow problems and finding sets of activities that have the optimal contribution to the resource level. These sets are then shown to compute an envelope. Finally, we describe a simple envelope algorithm and its complexity, and conclude discussing future work.

2 Resource Envelopes

Figure 1 shows an activity network with resource allocations. Our notation is equivalent to that of previous work on flexible scheduling with multiple-capacity resources [5][10]. The network has two time variables per activity, a start event and an end event (e.g., $\mathbf{e_{1s}}$ and $\mathbf{e_{1e}}$ for activity $\mathbf{A_1}$), a non-negative flexible activity duration link (e.g., **[2, 5]** for activity $\mathbf{A_1}$), and flexible

separation links between events (e.g., **[0, 4]** from $\mathbf{e_{3e}}$ to $\mathbf{e_{4s}}$). A time origin, $\mathbf{T_s}$, corresponds to time **0** and supports separation links to other events. Without loss of generality we assume that all events occur after $\mathbf{T_s}$ and before an event $\mathbf{T_e}$ rigidly connected to $\mathbf{T_s}$. The interval $\mathbf{T_sT_e}$ is the *time horizon* **T** of the network.

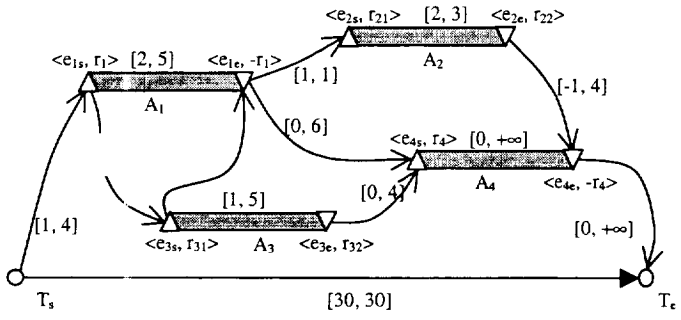

Figure 1: An activity network with resource allocations

Time origin, events and links constitute a Simple Temporal Network (STN) [8]. Unlike regular STNs, however, each event has an associated *allocation variable* with real domain (e.g., $\mathbf{r_{31}}$ for event $\mathbf{e_{3s}}$) representing the amount of resource allocated when the event occurs. We call this augmented network **R** a piecewise-constant Resource allocation STN (cR-STN). In the following we will assume that all allocations refer to a single, multi-capacity resource. The extension of the results to the case of multiple resources is straightforward. An event $\mathbf{e^-}$ with negative allocation is a *consumer,* while an $\mathbf{e^+}$ with positive allocation is a *producer.*

This formalization covers many of the usual models for resource allocations. For example, note that an event can be either a consumer or a producer in different instantiations of the allocation variables (e.g., event $\mathbf{e_{2s}}$ for which the bound for $\mathbf{r_{21}}$ is **[-1, 3]**). This allows reasoning about dual-use activities (e.g., the activities of starting a car and running it make use of the alternator as either a power consumer or producer). Moreover, some events can have resource allocations that are opposite to each other (e.g., $\mathbf{e_{1e}}$ vs. $\mathbf{e_{1s}}$). This allows modeling allocations that last only during an activity's occurrence, such as power consumption. Note, however, that this model does not cover continuous accumulation such as change of energy stored in a battery over time. A conservative approximation can be achieved by accounting for the entire resource usage at the activity start or end.

We will always assume that the cR-STN is temporally consistent. From the STN theory [8], this means that the shortest-path problem associated with **R** has a solution. Given two events $\mathbf{e_1}$ and $\mathbf{e_2}$ we denote with $\mathbf{|e_1e_2|}$ the shortest-path from $\mathbf{e_1}$ to $\mathbf{e_2}$. We will call a full instantiation of the time variables in **R** a *schedule* **s(.)** where **s(e)** is the time of occurrence of event **e** according to schedule **s.** The set **S** contains all possible consistent schedules for **R.** Each event **e** has a time bound **[et(e), lt(e)],** with **et(e)** and **lt(e)** respectively the earliest and latest time for **e.** The time bound represents the range of possible time values **s(e)** for all $\mathbf{s \in S}$. From the STN theory, we know that $\mathbf{et(e) = -|eT_s|}$ and $\mathbf{lt(e) = |T_se|}$. Finally, given three events, $\mathbf{e_1}$, $\mathbf{e_2}$ and $\mathbf{e_3}$, the triangular inequality among shortest paths $\mathbf{|e_1e_3| \leq |e_1e_2| + |e_2e_3|}$ holds.

A fundamental data structure used in the rest of the paper is the *anti-precedence graph,* **Aprec(R),** for a cR-STN **R.** The anti-precedence graph is similar to the precedence graph of [10] with the following differences: a) the links are in reverse order; 2) it does not distinguish between the set of strict precedence edges, $\mathbf{E_<}$, and the set of precedence edges that allow time equality, $\mathbf{E_\leq}$; and 3) several possible kinds of precedence graphs are allowable for a network **R.** Formally, **Aprec(R)** is a graph with the same events as **R** and such that for any two events

$\mathbf{e_1}$ and $\mathbf{e_2}$ with $|\mathbf{e_1}\,\mathbf{e_2}| \leq \mathbf{0}$ there is a path from $\mathbf{e_1}$ to $\mathbf{e_2}$ in **Aprec(R)**. Alternatively, we can say that an event $\mathbf{e_1}$ precedes another $\mathbf{e_2}$ in the anti-precedence graph if $\mathbf{e_1}$ cannot be executed before $\mathbf{e_2}$. A way to build an anti-precedence graph is to run an all-pairs shortest-path algorithm on **R** and retain only the edges with non-positive shortest distance. Smaller graphs can also be obtained by eliminating dominated edges. The choice of precedence graph type may affect performance but not the correctness of the algorithm described here.

The cost of computing **Aprec(R)** is therefore bound by the cost of computing the all-pairs shortest path graph for **R**, i.e., it is $\mathbf{O(NE + N^2 \lg V)}$ where **N** is the number of events (at most twice the number of activities in the plan) and **E** is the number of temporal distance constraints in the original cR-STN [7].

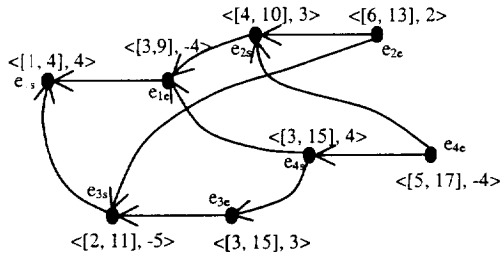

Figure 2: Anti-precedence graph with time/resource usage

Figure 2 depicts one of the anti-precedence graphs of the network in Figure 1 with the time bound and the maximum allowed resource allocation labeling each event.

3 Resource Contention Measures

Safe execution of a flexible activity networks needs to avoid resource contention, i.e., the possibility that for some consistent time assignment to the events there is at least one time at which the total amount of resource allocated is outside the availability bounds. There are essentially two methods for estimating resource contention: heuristic and exact. Most of the heuristic techniques [14] [12] [3] measure the probability of an activity requesting a resource at a certain time. This probability is estimated either analytically on a relaxed constraint network or stochastically by sampling time assignments on the full constraint network. The occurrence probabilities are then combined in an aggregate demand on resources over time, the contention measure. Probabilistic contention can give a measure of likelihood of a conflict. However, it is not a safe measure, i.e., the lack of detected conflicts does not exclude that a variable instantiation for the cR-STN could cause an inconsistent resource allocation.

Exact methods avoid this problem. They compute sufficient conditions for the lack of contention. [10] has a good survey of such methods. Current exact methods operate on relaxations of the constraint network. For example, edge-finding techniques [13] analyze how an activity can be scheduled relative to a subset of activities, comparing the sum of all durations with a time interval derived from the time bounds of all the activities under consideration. Relying solely on time bounds ignores much of the inter-activity constraints and is effective only when the time bounds are relatively tight. Therefore algorithms using these contention measures tend to eliminate much of the flexibility in the activity network. Some recent work [5] [10] goes further in exploiting inter-activity constraints. For example, [10] proposes a *balance constraint* that is based on conservative upper and lower bounds on the resource level immediately before and after each event **e**. These bounds precisely estimate the contribution of events that must precede **e** and overestimate the contribution of events that may or may not precede **e**. The over-estimate assumes that only the events with the worst contribution (producers for upper bounds and consumers for lower bounds) happen before **e**. The balance constraint appears to

work well in a flexible scheduling algorithm [10] but the bounds on which it is based may be very loose for networks with significant amounts of parallelism.

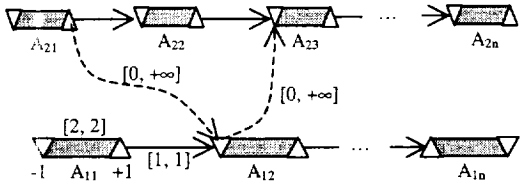

Figure 3: Over-constraining a flexible activity network

For example, consider the activity graph in Figure3, consisting of two rigid chains of **n** activities with the same fixed duration and the same fixed activity separation. Assume that the horizon **T** is wide enough to allow any feasible ordering among them. Each activity consumes one unit during its occurrence and the resource has two available units of capacity over time. It is clear that the maximum resource level requested by the flexible plan at any time cannot exceed two, and therefore all schedules obtained by merging any schedule of the two chains are feasible. A scheduling algorithm using a resource-level envelope will therefore recognize that the initial problem is already a feasible flexible plan and will terminate immediately. In contrast, a scheduler using the balance constraint will always detect an over-allocation until it somehow constrains the network (e.g., by systematic or local search) with constraints that are at least as tight as the two following cases: a) the start activity **n** of one chain occurs no later than the start of the second activity of the other; or b) more than two activities overlap and there is an activity **k** on one chain that must start between the end of activity **i** and the start of activity **i+2** on the other chain. The dashed arrows in Figure 3 represent the constraints posted in case b. The balance constraint cannot correctly handle this situation because it cannot account for the constraint structure of all possible parallel chains *simultaneously*. The rest of this paper shows that the full constraint structure can be efficiently exploited in calculating the resource-level envelope.

4 Resource Envelopes

As discussed in the introduction, we are seeking the maximum and minimum possible resource production (consumption) among all possible schedules of **R.** Note that the maximum (minimum) overall resource level induced by **R** for any possible schedule can always be obtained by assigning each allocation variable to its maximum (minimum) possible value. For any specific value assignment to the allocation variables, each event has a constant weight: positive, $\mathbf{c(e^+)}$, for a producer and negative, $\mathbf{-c(e^-)}$, for a consumer. Given a schedule $s \in S$ and a time $t \in T$, $\mathbf{E_s(t)}$ is the set of events **e** such that $\mathbf{s(e) \leq t}$. For any subset **A** of the set of events in **R**, **E(R)**, we define the *resource-level increment* as $\Delta(\mathbf{A}) = \mathbf{0}$ if $\mathbf{A} = \emptyset$, and $\Delta(\mathbf{A}) = \Sigma_{e^+, e^- \in A}\, c(e^+) - c(e^-)$ if $\mathbf{A} \neq \emptyset$. The following functions of time rigorously define the resource-level envelope:

- *Resource level* due to schedule s: $\mathbf{L_s(t) = \Delta(E_s(t))}$.
- *Maximum resource envelope* : $\mathbf{L_{max}(t) = \max_{s \in S}(L_s(t))}$.
- *Minimum resource envelope*: $\mathbf{L_{min}(t) = \min_{s \in S}(L_s(t))}$.

The *resource level envelope* that we seek is the interval bound function of time $\mathbf{[L_{min}(t), L_{max}(t)]}$. Since the methods to compute $\mathbf{L_{min}}$ and $\mathbf{L_{max}}$ can be obtained from each other with obvious term substitutions, we only develop the algorithm for $\mathbf{L_{max}}$.

Before we formally discuss our algorithm, we want to introduce some examples to give an intuitive feel of the foundations of the method. First consider an activity network consisting of

a single activity that produces or consumes resource capacity during its occurrence (Figure 4(a)). We could build $\mathbf{L_{max}}$ by asking at each time $\mathbf{t \in T}$ whether $\mathbf{A_1}$ can happen before, after or can overlap **t**. If the activity starts with a resource production (Figure 4(b)), then the resource level will be maximum at **t** if $\mathbf{A_1}$ starts, contains or ends at **t**. This is always possible between $\mathbf{et(e_{1s})}$ and $\mathbf{lt(e_{1e})}$. Within this interval $\mathbf{L_{max}(t) = 1}$, while outside $\mathbf{L_{max}(t) = 0}$. Conversely, if $\mathbf{A_1}$ starts with a consumer (Figure 4(c)), then the maximum resource level can be zero at time **t** only if $\mathbf{A_1}$ can start after **t** or can end before **t** for some schedule. This is possible only before $\mathbf{lt(e_{1s})}$ and after $\mathbf{et(e_{1e})}$. Therefore, $\mathbf{L_{max}(t) = -1}$ between $\mathbf{lt(e_{1s})}$ and $\mathbf{et(e_{1e})}$, and $\mathbf{L_{max}(t) = 0}$ everywhere else. This example suggests a strategy for computing $\mathbf{L_{max}}$ that looks at each event and considers the incremental contribution of the event's weight to the maximum resource envelope at the earliest time for producers or at the latest time for consumers.

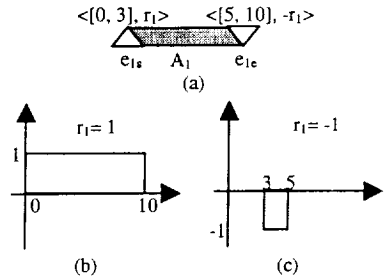

Figure 4: Maximum resource-level envelope for a single activity

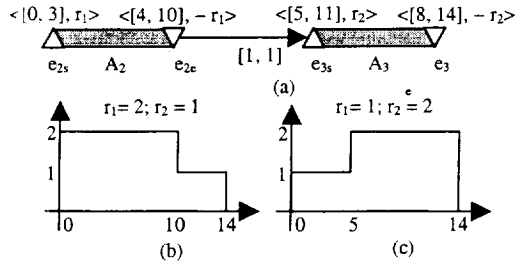

Figure 5: Maximum level envelope for two chained activities

For a complex network, however, this simple strategy is insufficient. Consider a rigidly linked pair of activities with a reusable resource allocation (Figure 5(a)). In this case, the time of occurrence of $\mathbf{e_{2e}}$ and $\mathbf{e_{3s}}$ are bound together. Looking at the contribution to the envelope of each event in isolation, we would want to add the contribution of $\mathbf{e_{2e}}$ as late as possible since it is a consumer, and the contribution of $\mathbf{e_{3s}}$ as early as possible, since it is a producer. The decision of which time to choose depends on the total contribution of *both* events. The total contribution will be added at $\mathbf{lt(e_{2e})}$ if the total contribution is a consumption (Figure 5(b)) or at $\mathbf{et(e_{3s})}$ if the total contribution is a production (Figure 5(c)). Note that in both cases $\mathbf{e_{2s}}$ and $\mathbf{e_{3s}}$ are *pending* at the selected time, i.e., their contribution has not been added yet to the envelope but they both could occur at the selected time. This suggests revising the strategy for computing $\mathbf{L_{max}}$ as follows: at the earliest or latest time of each event, select the set of pending events whose resource-level increment is maximum, eliminate these events from the pending events and declare the increment for $\mathbf{L_{max}}$ at that time to be their resource-level increment.

Now consider the network in Figure 1 and the event time bounds, maximum resource allocation and precedence graph in Figure 2. Assume that we want to compute $\mathbf{L_{max}(3)}$. The set of events that may be scheduled before, at or after time **3** is $\mathbf{\{e_{1s}, e_{1e}, e_{3s}, e_{3e}, e_{4s}\}}$. However, of these only $\mathbf{\{e_{1e}, e_{3s}, e_{3e}, e_{4s}\}}$ are pending since the contribution to $\mathbf{L_{max}}$ of $\mathbf{e_{1s}}$ occurs at its earliest time **1**. The subset of pending events that we need to consider at time **3** to compute the

increment on $\mathbf{L_{max}}$ are all those that are forced to occur at or before 3 assuming that some pending event occurs at 3. These subsets are $\{e_{1e}\}$, $\{e_{3s}\}$, $\{e_{3s}, e_{3e}\}$ and $\{e_{1e}, e_{3s}, e_{3e}, e_{4s}\}$. Unfortunately, each of these subsets has a negative weight and therefore none of them is selected at time 3 since, from the definition of resource-level increment, the empty event set has the maximum (zero) increment. At time 4 the set of pending events is augmented with e_{2s} and a new subset of pending events with positive weight, $\{e_{1e}, e_{2s}, e_{3s}, e_{3e}, e_{4s}\}$, is possible. This generates the increment that, added to $\mathbf{L_{max}(3)}$, gives us $\mathbf{L_{max}(4)}$.
The selection of the pending events subset of maximum resource-level increment is the key source of complexity in calculating $\mathbf{L_{max}}$. An exhaustive enumeration of all subsets is intractable in the general case. Fortunately, it turns out that this selection problem is equivalent to a maximum flow problem for an appropriate flow network derived from **Aprec(R)**. We discuss this rigorously in the rest of the paper.

5 Calculating Maximum Resource-Level Increments

Consider an interval $\mathbf{H \subseteq T}$. We can partition all events in **R** into three sets depending on their relative position with respect to **H**: 1) the *closed events* $\mathbf{C_H}$ with all events that must occur strictly before or at the start of **H**, i.e., such that that $\mathbf{lt(e) \leq start(H)}$; 2) the *pending events* $\mathbf{R_H}$ with all events that can occur within or at the end of interval **H**, i.e., such that $\mathbf{lt(e) > start(H)}$ and $\mathbf{et(e) \leq end(H)}$; and 3) the *open events* $\mathbf{O_H}$ with all events that must occur strictly after **H**, i.e., such that $\mathbf{et(e) > end(H)}$.
The set $\mathbf{R_H}$ could contain events that can be scheduled both inside and outside **H**. If **H=T**, then $\mathbf{C_T = \emptyset}$, $\mathbf{R_T = E(R)}$ and $\mathbf{O_H = \emptyset}$. If **H** is a single instant of time, i.e., **H=[t, t]**, we will use the simplifying notation $\mathbf{C_t = C_{[t,\,t]}}$, $\mathbf{R_t = R_{[t,\,t]}}$ and $\mathbf{O_t = O_{[t,\,t]}}$.
Assume that we want to compute the resource-level increment for a schedule **s** at a time $\mathbf{t \in H}$. This will always include the contribution of all events in $\mathbf{C_H}$ and none of those in $\mathbf{O_H}$ irrespective of **s** and **t**. With respect to the events in $\mathbf{R_H}$, if an event is scheduled to occur at or before **t**, then all of its predecessors (according to **Aprec(R)**) will also have to occur at or before **t**. In other words, it is possible to find a set of events $\mathbf{X \in R_H}$ such that the events $\mathbf{e_p \in R_H}$ that are scheduled no later than **t** in **s** are those in **X** or those $\mathbf{e_p \in R_H}$ such that $\mathbf{|e_x e_p| \leq 0}$ for some $\mathbf{e_x \in X}$. We call this the *predecessor set* $\mathbf{P_X}$ of **X**. Therefore, the resource level at time **t** for a given schedule **s** is the sum of the weights of events in $\mathbf{C_H}$ and in $\mathbf{P_X \subseteq R_H}$. Since we are trying to maximize the resource level, we will look for the sets $\mathbf{P_X}$ with the maximum total weight.
An important property that we will exploit later is that given two predecessor sets $\mathbf{P_X}$ and $\mathbf{P_Y}$, $\mathbf{P_X \cap P_Y}$ and $\mathbf{P_X \cup P_Y}$ are also predecessor sets.

5.1 Resource-Level Increments and Maximum Flow

We know that to compute $\mathbf{L_{max}}$ we want to look for sets $\mathbf{P_X}$ with maximum weight. We find these by computing a maximum flow for an auxiliary flow network built from $\mathbf{R_H}$ and **Aprec(R)**. For a complete discussion of maximum flow problems see [7]. Here we only highlight some concepts and relations that we will use.
First let us define the auxiliary flow problem that we will us to compute $\mathbf{L_{max}}$.
Resource Increment Flow Problem: *Given a set of pending events R_H for a cR-STN **R**, we define the resource increment flow problem $F(R_H)$ with source σ and sink τ as follows:*

1. *For each event $e \in R_H$ there is a corresponding node $e \in F(R_H)$.*
2. *For each event $e^+ \in R_H$, there is an edge $\sigma \rightarrow e^+$ with capacity $c(e^+)$.*
3. *For each event $e^- \in R_H$, there is an edge $e^- \rightarrow \tau$ with capacity $c(e^-)$, i.e., the opposite of e^-'s weight in **R**.*
4. *For each pair of e_1 and e_2 with an edge $e_1 \rightarrow e_2$ in the anti-precedence graph **Aprec(R)**, there is a corresponding link $e_1 \rightarrow e_2$ in $F(R_H)$ with capacity $+\infty$.*

Figure 6 shows the auxiliary flow problem for the anti-precedence graph in Figure 2, with every edge labeled with its capacity.

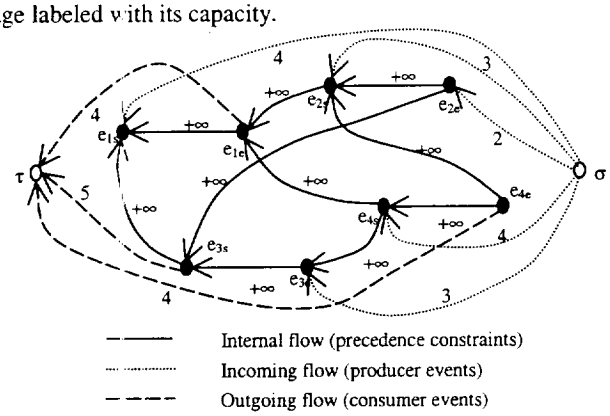

Figure 6: Resource increment flow problem

Consistent with the theory of maximum flows, we will indicate $\mathbf{f(e_1, e_2)}$ as the flow associated to a link $\mathbf{e_1 \rightarrow e_2}$ in $\mathbf{F(R_H)}$. The flow function has, by definition, several properties. It is skew-symmetric, i.e., $\mathbf{f(e_2, e_1) = -f(e_1, e_2)}$. Each flow has to be not greater than the capacity of the link to which it is associated. For example, referring to the flow network in Figure 6, $\mathbf{f(\sigma, e_{2e}) \leq 2}$, while $\mathbf{f(e_{2e}, \sigma) \leq 0}$ since there is no edge from $\mathbf{e_{2e}}$ and σ, a situation equivalent to the capacity of the edge $\mathbf{e_{2e} \rightarrow \sigma}$ being zero. We also use the implicit summation notation $\mathbf{f(A, B)}$, where **B** and **A** are disjoint event sets in $\mathbf{F(R_H)}$, to indicate the flow $\mathbf{f(A, B) = \Sigma_{a \in A} \Sigma_{b \in B} f(a, b)}$. Consider now any subset of events $\mathbf{A \subseteq R_H}$ and let us call $\mathbf{A^c}$ the set of events $\mathbf{A^c = R_H - A}$. From the final property defining a flow function, flow conservation, we can obtain the following: $\mathbf{f(\{\sigma\}, A) = f(A, \{\tau\}) + f(A, A^c)}$. The total network flow is defined as $\mathbf{f(\{\sigma\}, R_H) = f(R_H, \{\tau\})}$. The maximum flow of a network is a flow function $\mathbf{f_{max}}$ such that the total network flow is maximum.

A fundamental concept in the theory of flows is the *residual network*. This is a graph with an edge for each pair of nodes in $\mathbf{F(R_H)}$ with positive *residual capacity*, i.e., the difference between edge capacity and flow. Each edge in the residual network has capacity equal to the residual capacity. For example, considering the network in Figure 6, assume that $\mathbf{f(e_{1e}, \tau) = 3}$ and $\mathbf{f(\sigma, e_{2e}) = 2}$. The residual network for that flow will have the following edges: $\mathbf{e_{1e} \rightarrow \tau}$ with capacity **1**, $\mathbf{\tau \rightarrow e_{1e}}$ with capacity **3**, and $\mathbf{e_{2e} \rightarrow \sigma}$ with capacity **2**. Also note that any residual network for any flow of $\mathbf{F(R_H)}$ will always have an edge of infinite capacity for each edge in the precedence graph $\mathbf{Aprec(R)}$. An *augmenting path* is a path connecting σ to τ in the residual network. The existence of an augmenting path indicates that additional flow can be pushed from σ to τ. Alternatively, the lack of an augmenting path indicates that a flow is maximum.

A resource-level increment $\mathbf{\Delta(A)}$ for an event set $\mathbf{A \subseteq R_H}$ is related to a flow in $\mathbf{F(R_H)}$ as follows. We define the producer weight in **A** as $\mathbf{c(A^+) = \Sigma_{e+ \in A} c(e^+)}$ and the consumer weight in **A** as $\mathbf{c(A^-) = \Sigma_{e- \in A} c(e^-)}$. We also define the *producer residual* in **A** for a flow **f** of $\mathbf{F(R_H)}$ as $\mathbf{r(A^+) = c(A^+) - f(\{\sigma\}, A)}$, i.e., the total residual capacity of the edges from σ to **A**, and the *consumer residual* in **A** as $\mathbf{r(A^-) = c(A^-) - f(A, \{\tau\})}$.

Lemma 1: $\mathbf{\Delta(A) = r(A^+) - r(A^-) + f(A, A^c)}$.

Proof: $\mathbf{\Delta(A) = c(A^+) - c(A^-) = (c(A^+) - f(\{\sigma\}, A)) - (c(A^-) - f(\{\sigma\}, A)) = r(A^+) - (c(A^-) - f(A, \{\tau\}) - f(A, A^c)) = r(A^+) - r(A^-) + f(A, A^c)$.□

We now focus on predecessor sets such as $\mathbf{P_X}$.

Lemma 2: *$f(P_X, P^c_X) \leq 0$. Moreover, $f(P_X, P^c_X)=0$ if and only if $f(e_1, e_2)=0$ for each $e_1 \in P^c_X$ and $e_2 \in P_X$.*
Proof: From the definition of predecessor set there is no edge $\mathbf{e_2 \rightarrow e_1}$ in $\mathbf{F(R_H)}$ with $\mathbf{e_1 \in P^c_X}$ and $\mathbf{e_2 \in P_X}$. Therefore, $\mathbf{f(e_2, e_1) \leq 0}$ and $\mathbf{f(P_X, P^c_X) \leq 0}$. The second condition can be demonstrated by observing that the sum of any number of non-positive numbers is **0** if and only if each number is **0**.□

Corollary 1: *$\Delta(P_X) \leq r(P_X^+) - r(P_X^-)$.*
Proof: Immediate from Lemma 1 and Lemma 2.

5.2 Maximum flows and maximum resource-level increments

We can now find the maximum resource-level increment set $\mathbf{P_{max} \subseteq R_H}$. Since $\mathbf{P_{max} = \varnothing}$ may be true, the maximum resource-level increment is always non-negative. The computation of the set uses a maximum flow $\mathbf{f_{max}}$ of $\mathbf{F(R_H)}$. We indicate with $\mathbf{r_{max}(A)}$ the producer/consumer residual of A computed for $\mathbf{f_{max}}$. The following fundamental theorem holds.

Theorem 1: *Given a partial plan R_H, consider the (possibly empty) set P_{max} of events that are reachable from the source σ in the residual network of some f_{max} of $F(R_H)$. P_{max} is the predecessor set with maximum $\Delta(P_{max}) \geq 0$.*
Proof: Assume that $\mathbf{r_{max}(e^+) = 0}$ for each $\mathbf{e^+ \in F(R_H)}$. In this case no event is reachable from σ in the residual network, thus $\mathbf{P_{max} = \varnothing}$ and $\mathbf{\Delta(P_{max}) = 0}$. From Corollary 1, for any predecessor set $\mathbf{P_X}$ it is $\mathbf{\Delta(P_X) \leq -r_{max}(P_X^-) \leq 0 = \Delta(P_{max})}$ and therefore $\mathbf{\Delta(P_{max})}$ is maximum.
Assume now $\mathbf{r_{max}(e^+) > 0}$ for some $\mathbf{e^+ \in F(R_H)}$, in which case $\mathbf{P_{max}}$ is not empty. The following three properties hold.

1. *P_{max} is a predecessor set.*
   If not, there will be an event $\mathbf{e_2 \notin P_{max}}$ such that $\mathbf{|e_1 e_2| \leq 0}$ for some event $\mathbf{e_1 \in P_{max}}$. From the definition of $\mathbf{Aprec(R)}$, however, we know that there must be a path in $\mathbf{Aprec(R)}$ from $\mathbf{e_1}$ to $\mathbf{e_2}$. Since this path will be present in $\mathbf{F(R_H)}$ with all links having infinite capacity, the path will also always be present in any residual network for any flow. Therefore there is a path in the residual network going from σ to $\mathbf{e_1}$ (by definition of $\mathbf{P_{max}}$) and then to $\mathbf{e_2}$. Therefore, $\mathbf{e_2 \in P_{max}}$, which is a contradiction.
2. *$r_{max}(P_{max}^-) = 0$.*
   If not, there will be an event $\mathbf{e^- \in P_{max}}$ such that $\mathbf{r_{max}(e^-) > 0}$. We can therefore build an augmenting path of $\mathbf{F(R_H)}$ as follows: 1) a path from σ to $\mathbf{e^-}$ with positive residual capacity which exists by definition of $\mathbf{P_{max}}$; and 2) an edge $\mathbf{e^- \rightarrow \tau}$ with positive residual capacity $\mathbf{r_{max}(e^-)}$. The existence of the augmenting path means that $\mathbf{f_{max}}$ is not a maximum flow, which is a contradiction.
3. *$f_{max}(P_{max}, P^c_{max}) = 0$*
   Since $\mathbf{P_{max}}$ is a predecessor set, from the proof of Lemma 2 we know that $\mathbf{f_{max}(P_{max}, P^c_{max}) \leq 0}$. If $\mathbf{f_{max}(P_{max}, P^c_{max}) < 0}$, then there is a pair of events $\mathbf{e_1 \in P_{max}}$ and $\mathbf{e_2 \in P^c_{max}}$ such that $\mathbf{f_{max}(e_1, e_2) < 0}$. This means that the residual capacity from $\mathbf{e_1}$ to $\mathbf{e_2}$ is positive and therefore there is an edge $\mathbf{e_1 \rightarrow e_2}$ in the residual network. But by definition of $\mathbf{P_{max}}$, this means that $\mathbf{e_2 \in P_{max}}$, which is a contradiction.

Applying the properties of $\mathbf{P_{max}}$ to Lemma 1, $\mathbf{\Delta(P_{max}) = r_{max}(P_{max}^+) - r_{max}(P_{max}^-) + f_{max}(P_{max}, P^c_{max}) = r_{max}(P_{max}^+) > 0}$.
To prove the maximality of $\mathbf{P_{max}}$, observe from Corollary 1 that a non-empty predecessor set $\mathbf{P_X}$ has $\mathbf{\Delta(P_X) > 0}$ only if $\mathbf{r_{max}(e^+) > 0}$ for some $\mathbf{e^+ \in P_X}$. It is easy to see that $\mathbf{P_X}$ is the set of events reachable in the residual graph from $\mathbf{P_X^+ \subseteq P_{max}^+}$ and that the properties at points 2 and 3 above also hold for $\mathbf{P_X}$. Therefore, $\mathbf{\Delta(P_X) = r_{max}(P_X^+) \leq r_{max}(P_{max}^+) = \Delta(P_{max})}$, which proves the maximality of $\mathbf{P_{max}}$.□

The construction of $\mathbf{P}_{max}$ discussed before does not guarantee its uniqueness since it depends on a specific maximum flow among potentially many for $\mathbf{F}(\mathbf{R}_H)$. The following theorem proves, however, that $\mathbf{P}_{max}$ is indeed unique for all maximum flows of $\mathbf{F}(\mathbf{R}_H)$ and that it contains the minimum number of events among all predecessor sets with maximum positive resource-level increment.

Theorem 3: *The predecessor set P_{max} with maximum resource-level increment $\Delta(P_{max})$ and with minimum number of events is unique across all maximum flows of $F(R_H)$.*
Proof: Consider two maximum flows $\mathbf{f}_{max,j}$ and $\mathbf{f}_{max,k}$ among all maximum flows of $\mathbf{F}(\mathbf{R}_H)$ and assume that they produce two distinct maximum resource-level increment predecessor sets, $\mathbf{P}_{max,j}$ and $\mathbf{P}_{max,k}$. From the maximality of their increment, it must be $\Delta(\mathbf{P}_{max,k}) = \Delta(\mathbf{P}_{max,j}) = \Delta_{max}$. We can rewrite one maximum predecessor set as $\mathbf{P}_{max,j} = \mathbf{P}_{j\cap k} \cup \mathbf{P}_{k-j}$ where $\mathbf{P}_{j\cap k} = \mathbf{P}_{max,j} \cap \mathbf{P}_{max,k}$ and $\mathbf{P}_{k-j} = \mathbf{P}_{max,k} - \mathbf{P}_{max,j}$. The hypothesis of distinction of $\mathbf{P}_{max,j}$ and $\mathbf{P}_{max,k}$ yields $\mathbf{P}_{k-j} \neq \emptyset$.
First we observe that $\Delta(\mathbf{P}_{j\cap k}) = \Delta_{max}$. If not, $\Delta(\mathbf{P}_{j-k}) > 0$ and $\Delta(\mathbf{P}_{k-j}) > 0$. But this means that $\mathbf{P}_{j\cup k} = \mathbf{P}_{max,j} \cup \mathbf{P}_{max,k}$ is a predecessor set such that $\Delta(\mathbf{P}_{j\cup k}) > \Delta_{max}$, which is a contradiction. Consider now $\mathbf{P}_{max,j}$ and let us call $\mathbf{r}_j(\mathbf{e})$ the residual $\mathbf{r}_{max}(\mathbf{e})$ computed in flow $\mathbf{f}_{max,j}$. Since $\Delta(\mathbf{P}_{j\cap k}) = \Delta_{max}$, it must be $\mathbf{r}_j(\mathbf{e}^+) = 0$ if $\mathbf{e}^+ \in \mathbf{P}_{j-k}$. Also, $\mathbf{r}_j(\mathbf{e}^-) = 0$ for each $\mathbf{e}^- \in \mathbf{P}_{max,j}$. From Lemma 1, $\Delta(\mathbf{P}_{j-k}) = \mathbf{f}_{max,j}(\mathbf{P}_{j-k}, \mathbf{P}^c_{j-k}) = 0$. From Lemma 2 it follows that $\mathbf{f}_{max,j}(\mathbf{P}_{j-k}, \mathbf{P}_{j\cap k}) = 0$. Hence, there cannot be a link in the residual network from an event in $\mathbf{P}_{j\cap k}$ to one in $\mathbf{P}_{j-k}$. Therefore, $\mathbf{e} \in \mathbf{P}_{j-k}$ is not reachable from σ in the residual network and $\mathbf{P}_{j-k} = \emptyset$. Since this is true for any pair $\langle \mathbf{j}, \mathbf{k} \rangle$, $\mathbf{P}_{max}$ is unique.
The same argument applied to $\mathbf{P}_{max}$ and $\mathbf{P}^{\emptyset} \subseteq \mathbf{P}_{max}$ proves the minimality of $\mathbf{P}_{max}$, where $\mathbf{P}^{\emptyset}$ is a predecessor set such that $\Delta(\mathbf{P}^{\emptyset}) = \Delta_{max}$.□

6 Building Resource Envelopes

So far we know that the resource level for a schedule $\mathbf{s}$ at time $\mathbf{t} \in \mathbf{H}$ is equal to $\mathbf{L}_s(\mathbf{t}) = \Delta(\mathbf{C}_H) + \Delta(\mathbf{P}_X)$ for some predecessor set $\mathbf{P}_X$. However, it is not immediately obvious that the converse also applies. Given any predecessor set $\mathbf{P}_X$, we want to be able to determine a time $\mathbf{t}_X \in \mathbf{H}$, the *separation time*, and a schedule $\mathbf{s}_X$, the *separation schedule*, such that all and only the events in $\mathbf{C}_H \cup \mathbf{P}_X$ are scheduled at or before time $\mathbf{t}_X$. The existence of a separation schedule and a separation time is not obvious because of the upper-bound constraints in the STN, i.e., the metric links between events that do not contribute to the construction of $\mathbf{Aprec}(\mathbf{R})$. If some event occurs too early with respect to $\mathbf{t}_X$, an upper-bound constraint may force some event to occur before time $\mathbf{t}_X$ even if it is not a successor in $\mathbf{Aprec}(\mathbf{R})$. We now show that indeed we can find a separation time and schedule for *any* $\mathbf{P}_X$ and therefore also for $\mathbf{P}_{max}$. For the latter, we show that $\mathbf{t}_X$ is one of the times at which the resource level is maximum over $\mathbf{H}$ for any schedule. This yields the maximum resource envelope $\mathbf{L}_{max}$ if $\mathbf{H} = [\mathbf{t}, \mathbf{t}]$ and we scan $\mathbf{t}$ over the horizon $\mathbf{T}$.

6.1 Latest events

First we find the events in $\mathbf{P}_X$ that will be scheduled at time $\mathbf{t}_X$. We say that $\mathbf{e}$ is a *latest event* of $\mathbf{P}_X$ if it is not a strict predecessor of any other event in $\mathbf{P}_X$, i.e., for any $\mathbf{e}' \in \mathbf{P}_X$, $|\mathbf{e}'\ \mathbf{e}| \geq 0$. We will call $\mathbf{P}_{X,late}$ the set of all latest events in $\mathbf{P}_X$. Also, we define $\mathbf{P}_{X,early} = \mathbf{P}_X - \mathbf{P}_{X,late}$.

The following property holds between events in $\mathbf{P}_{X,late}$ and $\mathbf{P}_{X,early}$.

Property 1: *Any event $e_1 \in P_{X,early}$ is a strict predecessor of some event $e_2 \in P_{X,late}$, i.e., $|e_2 e_1| < 0$.*

Proof: Since $e_1 \in P_{X,early}$, there must be an event $e_{11} \in P_X$ such that $|e_{11}e_1| < 0$. If $e_{11} \in P_{X,late}$, the property is proven. Otherwise, we can find a finite chain of events $e_2 \rightarrow e_{1k} \rightarrow \ldots \rightarrow e_{11} \rightarrow e_1$ with $e_2 \in P_{X,late}$ and $|e_2e_{1k}| < 0$, $|e_{1j}e_{1\,j-1}| < 0$ and $|e_{11}e_1| < 0$, yielding $|e_2e_1| < 0$ for the triangular inequality of the shortest paths. If we could not find an $e_2 \in P_{X,late}$ to start such a finite chain, the chain would have to become a cycle of events in $P_{X,early}$, which contradicts the temporal consistency of **R**.□

6.2 Separation Time for Latest Events

We can construct a separation time t_X at which we will schedule all latest events.

Lemma 3: *There is a time interval* $[t_{X,min}, t_{X,max}]$ *that intersects all time bounds* $[et(e), lt(e)]$ *with* $e \in P_{X,late}$ *and such that* $start(H) \leq t_{X,max}$.
Proof: There must be a time value in common among all time bounds in $P_{X,late}$. If not, there would be two events $e_1, e_2 \in P_{X,late}$ such that $et(e_1) > lt(e_2)$ and, from the triangular inequality, $|e_1e_2| \leq -et(e_1) + lt(e_2) < 0$, which is inconsistent with the definition of $P_{X,late}$. Observe that there must be an event $e \in P_{X,late}$ such that $lt(e) = t_{X,max}$. If $start(H) > t_{X,max}$, then $lt(e) < start(H)$, which contradicts $e \in R_H$.□

We define the separation time as $t_X = \max(start(H), t_{X,min})$, with $t_X = start(H)$ if $P_X = \emptyset$. We can then show that each event in P^c_X can be scheduled after t_X.

Lemma 4: *For any event* $e \in P^c_X$, $lt(e) > t_X$
Proof: By definition of R_H it must be $lt(e) > start(H)$. So we only need to consider the case in which $t_X = t_{X,min} > start(H)$. In this case there is at least one event $e_1 \in P_{X,late}$ such that $et(e_1) = t_{X,min}$. For this event it is $|e_1 e| \leq -et(e_1) + lt(e)$. Since $e \in P^c_X$, it must be that $|e_1 e| > 0$, otherwise **e** would follow in $Apred(R)$ an event in P_X. Therefore, $lt(e) \geq et(e_1) + |e_1 e| > et(e_1) = t_{X,min}$.□

6.3 Separation schedule for predecessors

We now build the separation schedule s_X for P_X and t_X, i.e., a schedule such that $s_X(e) \leq t_X$ for $e \in C_H \cup P_X$ and $s_X(e) > t_X$ for $e \in P^c_X \cup O_X$. Note that the following discussion holds also if $P_X = \emptyset$ and $t_X = start(H)$.
The following algorithm builds the separation schedule.

1. Schedule all $e \in P_{X,late}$ at t_X, i.e., $s_X(e) = t_X$.
2. Propagate time through **R** obtaining new time bounds $[et'(e), lt'(e)]$ for each $e \in E(R)$.
3. Schedule all events $e \in E(R) - P_{X,late}$ at their new latest time, i.e., $s_X(e) = lt'(e)$.

For s_X to be a schedule, it must be consistent with respect to **R**. We see that step 1 is consistent since: 1) t_X belongs to the intersection of all latest event time bounds; and 2) since for any pair of latest events $|e_1e_2| \geq 0$, scheduling one at t_X does not prevent any other latest events to be also scheduled at time t_X. Step 3 above is also consistent because it is always possible to schedule all events at their latest times without temporal repropagation.
Now we need to show that the property defining a separation schedule is satisfied for s_X. Note that we already know that it is satisfied for events in $P_{X,late}$. By definition it is also satisfied for events in C_H and O_H. Therefore, we need to show that it is satisfied for $P_{X,early}$ and P^c_X.
a) $lt'(e) \leq t_X$ for all $e \in P_{X,early}$
According to Property 1 we can pick an event $e_1 \in P_{X,late}$ such that $|e_1 e| < 0$. From the triangular inequality we have $lt'(e) \leq lt'(e_1) + |e_1 e| < lt'(e_1) = t_X$.

b) $lt'(e) > t_X$ for all $e \in P^c_X$.

From Lemma 4 we know that before temporal repropagation it was $lt(e) > t_X$. After it, either $lt'(e)=lt(e)$, in which case the condition is satisfied, or $lt'(e)$ has changed due to a propagation that starts from some event $e_1 \in P_{X,\,late}$. So it must be $lt'(e) = t_X + |e_1 e|$. Since $e \in P^c_X$, it must be $|e_1 e| > 0$, otherwise e would follow in $Apred(R)$ an event in P_X. Hence, $lt'(e) > t_X$.

We can now compute the maximum resource level for any schedule within the inverval H. In the following, we indicate with $P_{max}(R_H)$ the P_{max} computed over $F(R_H)$.

Theorem 4: *The maximum resource level for any schedule of R over an interval $H \subseteq T$ is given by $\Delta(C_H) + \Delta(P_{max}(R_H))$.*

Proof: We know that at any time $t \in H$ the events in R_H that are scheduled before t are a predecessor set P_X. For the resource level at time t it is always $\Delta(C_H) + \Delta(P_X) \leq \Delta(C_H) + \Delta(P_{max}(R_H))$, the latter being the resource level at the separation time t_X for the separation schedule s_X.□

There are two interesting special cases of Theorem 4.

Corollary 2: *The maximum possible resource consumption for R over T is equal to $\Delta(P_{max}(R_T))$.*

This means that estimating the maximum possible resource consumption for a flexible plan over the entire time horizon has the same complexity as a maximum flow problem.

Corollary 3: $L_{max}(t) = \Delta(C_t) + \Delta(P_{max}(R_t))$.

The last formula tells us how to compute the resource-level envelope at a specific time. We now need to find an efficient algorithm to compute the resource-level envelope over the entire horizon T.

7 Efficient Computation of Resource Envelopes

From Corollary 3, the naïve approach to compute a resource-level envelope would be to iterate over all possible $t \in T$. However, we only need to compute L_{max} at times when either C_t or R_t changes. This can only happen at $et(e)$ or $lt(e)$ for any $e \in E(R)$. Therefore we need to compute new levels for L_{max} only $2N$ times, where N is the number of start/end events in the original activity network. For each such computation, we need to: a) compute $P_{max}(R_t)$ by running a maximum flow on a network with at most N nodes; and 2) collect and sum the events in C_t and $P_{max}(R_t)$. The total complexity of the algorithm is therefore $O(N\ O(maxflow(N)) + N^2)$, where $O(maxflow(N))$ is the complexity of finding a maximum flow with an arbitrary maximum flow algorithm. For modern algorithms using the "preflow push" method [9], the worst case complexity can be $O(N^3)$. Extensive empirical studies show that the practical complexity of variations of the method can be as fast as $O(N^{1.5})$ [1]. This suggests that resource-level envelopes could operate in the inner loop of scheduling search algorithms, especially if they can be computated incrementally

8 Conclusions

In this paper we describe an efficient algorithm to compute the tightest exact bound on the resource level induced by a flexible activity plan. This can potentially save exponential

amounts of work with respect to currently available looser bounds. Future work will pursue two directions. The first is developing more incremental algorithms for the computation of the envelope. Using a temporal scanning of the events in the temporal network, it should be possible to significantly reduce the size of the networks on which the maximum flow algorithm needs to be run. This could significantly speed up the envelope calculation. The second direction will test the practical effectiveness of resource envelopes in the inner loop of search algorithms for multi-capacity resource scheduling, such as those used in (Laborie, 2001). This includes inner-loop backtracking and termination tests and variable and value ordering heuristics that exploit more directly the properties of the resource envelopes.

Acknowledgements

Ari Jonsson and Jeremy Frank were instrumental in pushing me to focus on this problem. During a dinner discussion at possibly the worst tourist restaurant in Paris, Grigore Rosu convinced me that the key of the resource-level envelope problem lies with the maximum-flow problem. Paul Morris gave me several helpful comments and suggested a simplification of the proof of theorem 1. Finally, Amedeo Cesta, Mary Bernardine Dias, Gregory Dorais, Paul Tompkins, an anonymous reviewer of a previous, unsuccessful submission, and a reviewer of the current successful one, gave several comments that helped me improve the presentation.
This work was performed with the support of the Intelligent Systems project of the Computing, Information and Communication Technologies research program of the National Aeronautics and Space Administration.

References

1. R.K. Ahuja, M. Kodialam, A.K. Mishra, J.B. Orlin. Computational Investigations of Maximum Flow Algorithms, *European Journal of Operational Research*, Vol 97(3), 1997.
2. K.R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.
3. J.C. Beck, A.J. Davenport, E.D. Davis, M.S. Fox. Beyond Contention: Extending Texture-Based Scheduling Heuristics. in *Proceedings of AAAI 1997*, Providence, RI, 1997.
4. A., Cesta, A. Oddi, S.F. Smith, A Constraint-Based Method for Resource Constrained Project Scheduling with Time Windows, CMU RI Technical Report, February 2000.
5. A. Cesta, C. Stella. A Time and Resource Problem for Planning Architectures. *Proceedings of the 4th European Conference on Planning (ECP 97)*. Toulouse, France, 1997.
6. H. S.F. Cooper Jr., The Loneliness of the Long-Duration Astronaut, *Air & Space/Smithsonian*, June/July 1996, available at http://www.airspacemag.com/ASM/Mag/Index/1996/JJ/llda.html
7. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. Cambridge, MA, 1990.
8. R. Dechter, I. Meiri, J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61-95, May 1991.
9. A.V. Goldberg, R.E. Tarjan. A New Approach to the Maximum-Flow Problem. *Journal of the ACM*, Vol. 35(4), 1988.
10. P. Laborie, Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results, *Proceedings of ECP 2001*, Toledo, Spain, 2001.
11. P. Morris, N. Muscettola, T. Vidal. Dynamic Control of Plans with Temporal Uncertainty, in *Proceedings of IJCAI 2001*, Seattle, WA. 2001
12. N. Muscettola. On the Utility of Bottleneck Reasoning for Scheduling. in *Proceedings of AAAI 1994*, Seattle, WA, 1994.
13. W.P.M. Nuijten. Time and Resource Constrained Scheduling: a Constraint Satisfaction Approach. PhD Thesis, Eindhoven University of Technology, 1994.
14. N. Sadeh. Look-ahead techniques for micro-opportunistic job-shop scheduling. PhD Thesis, Carnegie Mellon University, CMU-CS-91-102, 1991.
15. M. Zweben, M.S. Fox. *Intelligent Scheduling*. Morgan Kaufmann, San Francisco, 1994.