Computing the envelope for stepwise constant resource allocations

Nicola Muscettola

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

Abstract

Estimating 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. The incremental solution of 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. The staged algorithm has the same computational complexity of solving a maximum flow problem on the entire flow network. This makes this method computationally feasible for use in the inner loop of search-based scheduling algorithms.

Introduction

Retaining flexibility in the execution of activity plans is a fundamental technique for dealing with the uncertain conditions under which the plans will be executed. For example, flexible plans allow explicit reasoning about the temporal uncontrollability of exogenous events (Morris, Muscettola, Vidal 2001) and the incorporation of execution countermeasures within the flexible network. Tightly constrained schedules (e.g., schedules that assign a precise start and end time to all activities) are typically brittle and it is very difficult to closely follow their directions during execution. For an example of what overly tight schedules can do to an intelligent execution system, consider the "Skylab strike" (Cooper, 1996), when during the Skylab 4 mission astronauts went on a sit-down strike after 45 days of trying to catch up with the demands of a fast paced schedule with no room for them to adjust to the space environment.

A major obstacle to building flexible schedules, however, remains the difficulty of accurately estimating the amount of resources that a flexible plan may need across all of its possible executions. This problem is particularly difficult for resources with multiple capacity that can be both consumed and produced. In the worst case large plans may exhibit both a high level of activity parallelism and a large number of required synchronization constraints among activities. Most of the scheduling methods available to date for this problem (Cesta, Oddi Smith, 2000) eventually produce a fixed activity schedule, even if they make substantial use of an activity plan's flexibility during schedule construction.

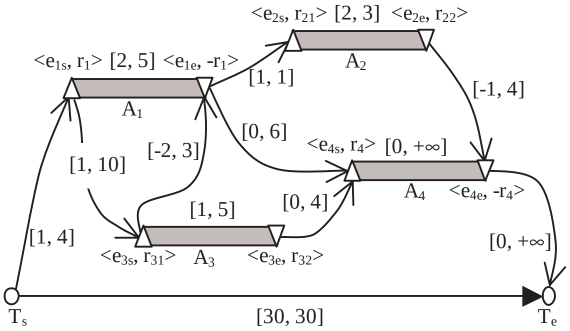
To appreciate the difficulty of precisely estimating resource consumption, consider the fact that a flexible activity plan has an exponential number of possible instantiated schedules. This means that methods based on complete enumeration are typically out of the question. Lately, however, new techniques have been developed (Laborie, 2001) based on direct propagation of information on the temporal constraints of the plan. This yields both an upper bound and a lower bound on the resource level required by the plan over time. This information can be used in various ways, e.g., to decide when to backtrack (when the lower/upper bound interval is outside of the range of allowed resource levels at some time) and when a solution has been achieved (when the lower/upper bound interval is inside the range of allowed resource levels at all times). Bound tightness is extremely important computationally since both as backtracking and termination criteria it can save a potentially exponential amount of search when compared to a looser bound.

A natural question is whether constructing the *tightest* possible resource level bounds is computationally feasible. This paper answers this question in the affirmative. We describe an efficient algorithm for the computation of a *resource level envelope*, a resource level bound such that for each time there exists at least a schedule for the activity plan that will consume the amount of resource indicated by the bound. The algorithm is polynomial, with complexity equivalent to solving a maximum flow problem on a flow network of the size of the original activity plan.

In the rest of the paper we first introduce 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 in resource level bounds is inadequate. Then we give an intuitive understanding of our method to compute the resource envelope. Then we establish the connection between maximum flow problems and finding sets of activities that have the optimal contribution to the resource envelope. We then show that these sets of activities compute an envelope. We then describe an efficient envelope algorithm and its complexity. We conclude discussing future work.

Activity Networks and Resource Consumption

Figure 1 shows an activity network with resource allocations. 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. 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.



$r_1 \in [1, 4]$ $\qquad$ $r_{31} \in [-7, -5]$
$r_{21} \in [-1, 3]$ $\qquad$ $r_{32} \in [1, 3]$
$r_{22} \in [1, 2]$ $\qquad$ $r_4 \in [2, 4]$

Figure 1: An activity network with resource allocations.

Time origin, events and links constitute a Simple Temporal Network (STN) (Dechter, Meiri, Pearl 1991). 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 will 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*.

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., starting a car and running it both make use of the alternator as a power consumer or producer). Moreover, some events can have opposite resource allocation of other events (e.g., $\mathbf{e_{1e}}$ vs. $\mathbf{e_{1s}}$). This allows modeling reusable allocations, such as power consumption by an activity. Note that this model does not cover continuous accumulation such as change of energy stored in a battery over time. A conservative approximation can however 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, this means that the shortest-path problem associated to **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.** We will call **S** the set of all possible consistent schedules for **R**. Each event **e** has a time bound **[et(e), lt(e)],** with $\mathbf{et(e)} = -|\mathbf{eT_s}|$ and $\mathbf{lt(e)} = |\mathbf{T_se}|$**,** representing the range of time values **s(e)** for all $\mathbf{s \in S}$**.** Finally, given three events, $\mathbf{e_1}$, $\mathbf{e_2}$ and $\mathbf{e_3}$, the triangular inequality $|\mathbf{e_1e_3}| \le |\mathbf{e_1e_2}| + |\mathbf{e_2e_3}|$ holds.

A fundamental data structure used in the rest of the paper is the *precedence graph,* **Pred(R),** for a cR-STN **R**. This is defined as a graph with the same events as **R** and such that for any two events $\mathbf{e_1}$ and $\mathbf{e_2}$ with $|\mathbf{e_1\ e_2}| \le \mathbf{0}$ there is a path from $\mathbf{e_1}$ to $\mathbf{e_2}$ in **Pred(R).** Alternatively, we can say that an event $\mathbf{e_1}$ precedes another $\mathbf{e_2}$ in the precedence graph if $\mathbf{e_1}$ cannot be executed before $\mathbf{e_2}$. There are several possible precedence graphs for a network **R.** A way to build one is to run an all-pairs shortest-path algorithm and retain only the edges with non-positive shortest distance. Smaller graphs can be obtained by eliminating dominated edges, e.g., by applying dispatchability minimization (Tsamardinos, Muscettola, Morris 1998). The cost of computing **Pred(R)** is bound by $\mathbf{O(VE + V^2\ lg\ V)}$ where **V** is the number of events and **E** the number of temporal distance constraints in the original cR-STN. The use of different precedence graphs may affect algorithm performance but does not affect the theoretical foundation described here.

Considering again the activity network in Figure 1, Figure 2 depicts one of its precedence graphs with each event labeled with the time bound and the maximum allowed resource allocation.

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 (Sadeh, 1991)(Muscettola, 1994) (Beck et al., 1997) 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 occurring. However, it is not a safe measure, i.e., the fact that it does not identify any conflict does not exclude the possibility that the cR-STN could have a variable instantiation with inconsistent resource allocation. Exact methods avoid this problem and are based on the computation of sufficient conditions for the lack of contention. (Laborie, 2001) has a good survey of such methods. Current exact methods operate on relaxations of the full constraint network. For example, edge-finding techniques (Nuijten, 1994) analyze how an activity can be scheduled relatively 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 only on time bounds ignores much of the inter-activity flexible constraints and tend to be effective only when the time bounds are relatively tight. Therefore algorithms that use these contention measures tend to eliminate much of the flexibility in the activity network. (Laborie, 2001) goes further in exploiting the information about mutual activity constraints. One of the two metrics proposed in that paper is the *balance constraint*, an event-centered approach that estimates upper and lower bounds on the resource level immediately before and after each event **e** in the cR-STN. These bounds precisely estimate the contribution of events that must precede **e** and overestimate the contribution of events that may or may not happen before **e.** The over-estimate is obtained considering only the worst-case situation in which only the events that have the worst contribution (producers for upper bounds and consumers for lower bounds) happen before **e**. Although the balance constraint better exploits the information in the activity network, the bounds that it produce may be very loose for networks with significant amounts of parallelism.

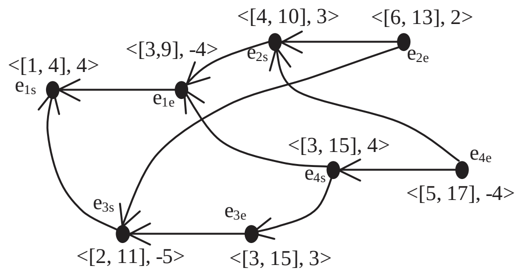


Figure 2: Precedence graph with time/resource usage.

For example, consider the activity graph in Figure 3, consisting of two rigid chains of **n** activities with the same fixed duration and the same fixed activity separation, and occurring on a horizon **T** wide enough to allow any feasible ordering among them. Each activity has a reusable consumption of one unit and the resource has two available units of capacity over time. It is clear that all the executions of this activity network are consistent with the resource constraint, since the maximum resource consumption is one unit of capacity for each chain at any time. However, the balance constraint will always detect an over-allocation unless the network is further constrained in one of two ways: 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 (Figure 3). These additional constraints unnecessarily eliminate a large number of legal executions of the activity network.

The cause of the inability of the balance constraint to correctly handle this situation depends on its inability to account for the constraint structure of parallel chains *simultaneously* since it can only take advantage of the full structure of the network for chains of predecessors. In this paper we will show that it is possible to effectively use *all* precedence constraints in the network simultaneously leading to the estimate of the best possible upper bound for resource consumption.

Resource Envelopes

Our approach is to build the tightest possible resource-centered exact contention measure. This means that for any possible time value we will compute the maximum and minimum possible 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. More formally, given a schedule $\mathbf{s \in S}$ and a time $\mathbf{t \in T}$, $\mathbf{E_s(t)}$ 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 will call the *resource level increment* $\mathbf{\Delta(A) = \Sigma_{e+,e- \in A}\ c(e^+) - c(e^-)}$. The increment is also defined for the empty set as $\mathbf{\Delta(\varnothing)=0}$. Therefore, the *resource level* at time **t** due to schedule **s** is $\mathbf{L_s(t) = \Delta(E_s(t))}$. The *maximum resource envelope* at time **t** is $\mathbf{L_{max}(t) = max_{s \in S}\ (L_s(t))}$. Similarly, the *minimum resource envelope* at time **t** is $\mathbf{L_{min}(t) = min_{s \in S}\ (L_s(t))}$. Our goal is to compute both $\mathbf{L_{max}}$ and $\mathbf{L_{min}}$ over **T**.

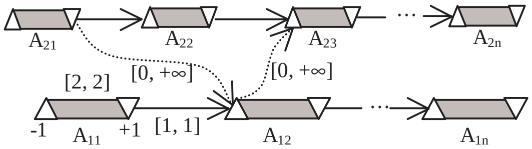


Figure 3: Over-constraining of activity network flexibility.

In the following sections we will rigorously develop an efficient algorithm to compute envelopes. Here, we want to give an intuitive account of what is involved and of the key complexity of envelope computation by analyzing some simple examples.

First consider a single activity with a reusable allocation (Figure 4(a)). We could build the envelope $\mathbf{L_{max}}$ 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 we want $\mathbf{A_1}$ to start, contain or end at **t**. This is always possible between $\mathbf{et(e_{1s})}$ and $\mathbf{lt(e_{2s})}$. During this interval the resource envelope is **1** while outside of it it is **0.** Conversely, if $\mathbf{A_1}$ starts with a consumer (Figure 4(c)), then we want for the activity to happen completely before or after **t**. This is possible only before $\mathbf{lt(e_{1s})}$ and after $\mathbf{et(e_{1e})}$. The envelope will be **0** at every time except between $\mathbf{lt(e_{1s})}$ and $\mathbf{et(e_{1e})}$ where it will be **–1.** This suggests a strategy that looks at each event and considers the incremental contribution of the event's weight to the envelope at the earliest time for producers or at the latest time for consumers.

When computing $\mathbf{L_{max}}$ for a complex network, however, events cannot usually be scheduled independently. Consider the simplest network, i.e., 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 on 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 a strategy that considers all pending events at either the earliest time or the latest time of some event and schedule those that either must be scheduled or are advantageous, i.e., contribute overall with a production of resource.

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 **L_{max}(3)**, the maximum envelope at time 3. The set of events that may be scheduled before, at or after time 3 is **{e_{1s}, e_{1e}, e_{3s}, e_{3e}, e_{4s}}**. However, of these only **{e_{1e}, e_{3s}, e_{3e}, e_{4s}}** are pending since it is advantageous to consider **e_{1s}** at its earliest time **1**. The subset of events that we could consider at time **3** are all those that will have to occur at or before **3** assuming that we select for some set of events to occur at **3**. These subsets are **{e_{3s}}, {e_{1s}}, {e_{3e}, e_{3s}}** and **{e_{4s}, e_{1e}, e_{3e}, e_{3s}}.** Unfortunately, each of these subsets has a negative weight and therefore none of them is considered at time **3.** At time **4** the set of pending events is augmented with **e_{2s}** and the total contribution of the new subset of pending events **{e_{2s}, e_{4s}, e_{1e}, e_{3e}, e_{3s}}** is positive.

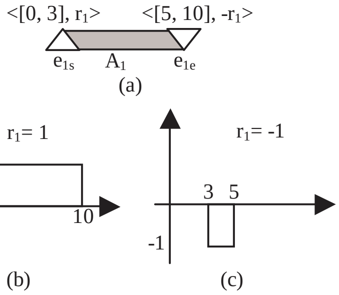


Figure 4: Maximum resource envelope for a single activity.

The selection of a maximally advantageous subset among the pending events is the key source of complexity of envelope calculation. An exhaustive enumeration of all subsets can obviously be very expensive. Fortunately we can make very good use of the information in the precedence graph. It turns out that this problem is equivalent to a maximum flow problem solved on an appropriate auxiliary flow network built on the basis of **Pred(R).** We will discuss this rigorously in the rest of the paper.

Calculating Maximum Resource Level Increments

Consider now an interval **H⊆T.** We can partition all events in **R** into three sets depending on their relative position with respect to **H**: 1) the *closed events* **C_H** with all events that must occur strictly before or at the start of **H,** i.e., such that that **lt(e) ≤ start(H);** 2) the *pending events* **R_H** with all events that can occur within or at the end of interval **H**, i.e., such that **lt(e) > start(H)** and **et(e) ≤ end(H)**; and 3) the *open events* **O_H** with all events that must occur strictly after **H**, i.e., such that **et(e) > end(H).**

The set **R_H** could contain events that can be scheduled both inside and outside **H**. If **H=T,** then **C_T= ∅, R_T = E(R)** and **O_H=∅.** The interval **H** could be reduced to a single instant of time, i.e., **H=[t, t].** In this case we will use the simplifying notation **C_t=$C_{[t,\ t]}$, R_t=$R_{[t,\ t]}$** and **O_t=$O_{[t,\ t]}$.**

We are interested in a particular kind of subset of **R_H.** Assume that we wanted to compute the resource level increment for a schedule **s** at a time **t∈H.** This will always include the contribution of all events in **C_H** and none of those in **O_H** irrespective of **s** and **t**. With respect to the events in **R_H,** we can see that if an event is scheduled to occur at or before **t** then all of its predecessors (according to **Pred(R)**) will also have to occur at or before **t**. In other words, it is possible to find a set of events **X ∈ R_H** such that the events **e_p∈R_H** that are scheduled no later than **t** in **s** are those such that **|$e_x e_p$| ≤ 0** for some **e_x ∈ X.** We call this the *predecessor set* of **X, P_X.** Therefore, the resource level at time **t** for a given schedule **s** is the sum of the weights of events in **C_H** and in **P_X.**

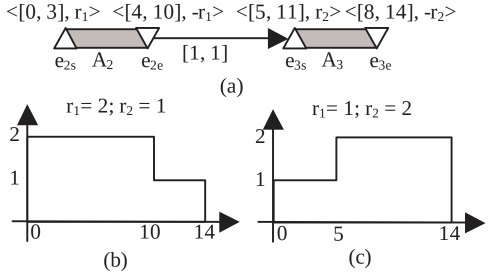


Figure 5: Maximum envelope for two chained activities.

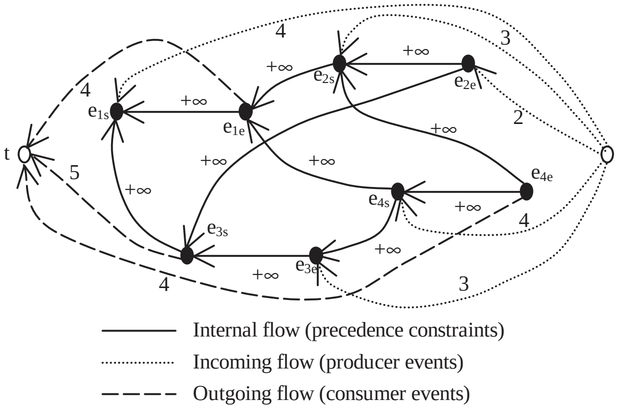
It is easy to verify that given two predecessor sets **P_X** and **P_Y**, both **P_X ∩ P_Y** and **P_X ∪P_Y** are also predecessor sets.

Resource Level Increments and Maximum Flow

Since we are interested in the maximum resource level we want to find the predecessor set with maximum resource level increment. We will do so by finding a maximum flow for an auxiliary flow network built from **R_H** and **Pred(R).**

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** ∈ **R_H** there is a corresponding node **e∈F(R_H).***
2. *For each event **e^+ ∈R_H,** there is an edge **σ→ e^+** with capacity **c(e^+)**.*
3. *For each event **e^- ∈ R_H,** there is an edge **e^-→τ** 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→e_2** in the precedence graph **Pred(R)**, there is a corresponding link **e_1→e_2** in **F(R_H)** with capacity +∞.*



4
3
+∞
e2s
e2e
4
+∞
e1s
e1e
+∞
2
t
s
+∞
+∞
+∞
e4e
5
e4s
+∞
+∞
e3s
4
+∞
e3e
+∞
4
3
Internal flow (precedence constraints)
Incoming flow (producer events)
Outgoing flow (consumer events)

Figure 6: Resource increment flow problem.

As an example, Figure 6 shows the auxiliary flow problem for $\mathbf{R_T}$ relative to the activity network in Figure 1.

A detailed discussion of flow problems is beyond the scope of this paper (for a complete treatment see (Cormen, Leiserson, Rivest 1990). Here we highlight some fundamental concepts and relations that we will use. We will indicate as $\mathbf{f(e_1, e_2)}$ the flow associated to a link $\mathbf{e_1 \rightarrow e_2}$ in $\mathbf{F(R_H)}$. The flow function 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, $\mathbf{0 \leq f(\sigma, e_{2e}) \leq 2}$, $\mathbf{0 \leq f(e_{1e}, \tau) \leq 4}$ and $\mathbf{f(e_{4e}, e_{4s}) \geq 0}$. Note that a flow from $\mathbf{e_1}$ to $\mathbf{e_2}$ can be negative only if the flow network contains an edge $\mathbf{e_2 \rightarrow e_1}$ with positive capacity. We also use an 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 $\underline{\mathbf{A}}$ the set of events $\mathbf{\underline{A} = R_H - A}$. The following flow balance constraint always holds: $\mathbf{f(\{\sigma\}, A) = f(A, \{\tau\}) + f(A, \underline{A})}$. The total network flow is defined as $\mathbf{f(\{\sigma\}, R_H) = f(R_H, \{\tau\})}$. The maximum flow of a network is a function $\mathbf{f_{max}}$ such that the total network flow is maximum.

The fundamental concept used by all known maximum flow algorithms is the *residual network*. This is a flow network with an edge for each pair of nodes in $\mathbf{F(R_H)}$ for which the *residual capacity*, i.e., the difference between edge capacity and flow, is positive. 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 flowing 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 **Pred(R).**

In this paper we will make use of three different kinds of paths in the residual network. The first is an *augmenting path* connecting $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$. The existence of an augmenting path indicates that additional flow can be pushed from $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$. Several maximum flow algorithms operate by searching for augmenting paths. Alternatively, the lack of an augmenting path is the condition that indicates that a flow is a maximum flow. The second kind of path is a *flow-shifting path*, a loop connecting $\boldsymbol{\tau}$ to τ which does not affect the overall flow in the network. Finally, the third kind is a *reducing path*, i.e., a path from $\boldsymbol{\tau}$ to $\boldsymbol{\sigma}$. Pushing flow through a reducing path reduces a network's flow.

We now establish the relation between the resource level increment $\mathbf{\Delta(A)}$ and any flow in $\mathbf{F(R_H)}$. 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** as $\mathbf{r(A^+) = c(A^+) - f(\{\sigma\}, A)}$, i.e., the total residual capacity of the edge incoming **A** from **s,** and the *consumer residual* in **A** as $\mathbf{r(A^-) = c(A^-) - f(A, \{\tau\})}$. The following relation holds.

Lemma 1: $\mathbf{\Delta(A) = r(A^+) - r(A^-) + f(A, \underline{A})}$.
Proof: $\mathbf{\Delta(A) = c(A^+) - c(A^-) = (c(A^+) - f(\{\sigma\}, A)) - (c (A^-) - f(\{\sigma\}, A)) = r(A^+) - (c(A^-) - f(A, \{\tau\}) \; - \; f(A, \underline{A})) = r(A^+) - r(A^-) + f(A, \underline{A})}$. □

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

Lemma 2: ***f(P_X, \underline{P}_X) ≤ 0.*** *Moreover,* $\boldsymbol{f(P_X, \underline{P}_X)=0}$ *if and only if* $\boldsymbol{f(e_1, e_2)=0}$ *for each* $\boldsymbol{e_1 \in \underline{P}_X}$ *and* $\boldsymbol{e_2 \in P_X}$.
Proof: From the definition of predecessor there is no edge $\mathbf{e_2 \rightarrow e_1}$ in $\mathbf{F(R_H)}$ with $\mathbf{e_1 \in \underline{P}_X}$ and $\mathbf{e_2 \in P_X}$. Therefore, $\mathbf{f(e_2, e_1) \leq 0}$ and $\mathbf{f(P_X, \underline{P}_X) \leq 0}$. 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**. □

Another way to express Lemma 2 is that $\mathbf{f(P_X, \underline{P}_X)=0}$ if and only if there is no link $\mathbf{e_1 \rightarrow e_2}$ in the residual network where $\mathbf{e_1 \in \underline{P}_X}$ and $\mathbf{e_2 \in P_X}$.

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

Maximum flows and maximum resource level increments

We are now ready to find the maximum positive resource level increment. Note that we are not interested in event sets with negative resource increments since, as we discussed before, we will only account for events in our resource envelope simulation if they have a positive contribution. If they do not, we will take them into account when we must, i.e., when their temporal upper bound becomes lower or equal to the current simulation time.

First we address the problem of whether $\mathbf{R_H}$ contains a set of predecessors $\mathbf{P^*}$ with positive resource level increment, i.e., $\mathbf{\Delta(P^*) > 0}$. To do so we will make use of a maximum flow $\mathbf{f_{max}}$ of $\mathbf{F(R_H)}$. We will indicate with $\mathbf{r_{max}(A)}$ producer/consumer residual computed for $\mathbf{f_{max}}$. The following fundamental theorem holds.

Theorem 1: *Given a partial plan* $\boldsymbol{R_H}$, *there is a predecessor set* $\boldsymbol{P^*}$ *such that* $\boldsymbol{\Delta(P^*)>0}$ *if and only if* $\boldsymbol{r_{max}(R_H{}^+) > 0}$.
Proof: □ **:** We prove that if there is a $\mathbf{P^*}$ such that $\mathbf{\Delta(P^*) > 0}$, then $\mathbf{r_{max}(R_H{}^+) > 0}$. Assume that $\mathbf{r_{max}(R_H{}^+) =} \; \boldsymbol{0}$. This means that for any predecessors subset $\mathbf{P_X}$ we have $\mathbf{r_{max}(P_X{}^+) = 0}$. From Lemma 2 we would have $\mathbf{\Delta(P^*) \leq -r_{max}(P^{*-}) \leq 0}$, that is a contradiction.
$\Leftarrow$**:** We prove that if $\mathbf{r_{max}(R_H{}^+) > 0}$ then we can identify a $\mathbf{P^*}$ such that $\Delta(\mathbf{P^*})\mathbf{>0}$. If $\mathbf{r_{max}(R_H{}^+)}$ is positive, there must be some $\mathbf{e^+}$ such that $\mathbf{r_{max}(e^+) > 0}$. Let's select $\mathbf{P^*}$ as the set of events reachable by some path in the residual network originating from $\mathbf{e^+}$. The following three properties hold.

1. ***P*** *is a predecessor set.*
   If not, there will be an event $\mathbf{e_2} \notin \mathbf{P^*}$ such that $\mathbf{|e_1 e_2| \leq 0}$ for some event $\mathbf{e_1 \in P^*}$. From the definition of **Pred(R),** however, we know that there must be a path in **Pred(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 $\mathbf{e^+}$ to $\mathbf{e_1}$ to $\mathbf{e_2}$ and $\mathbf{e_2 \in P^*}$, which is a contradiction.
2. $\boldsymbol{r_{max}(P^{*-}) = 0}$.
   If not, there will be an event $\mathbf{e^- \in P^*}$ such that $\mathbf{r_{max}(e^-) > 0}$. We can therefore build an augmenting path of $\mathbf{F(R_H)}$ as follows: 1) an edge $\boldsymbol{\sigma \rightarrow e^+}$ with positive residual capacity $\mathbf{r_{max}(e^+)}$; **2)** a path in the residual network from

e^+ to e^-, which exists by definition of P^*; and 3) an edge $e^- \rightarrow \tau$ with residual capacity $r_{max}(e^-)$. The existence of the augmenting path means that f_{max} is not a maximum flow, which is a contradiction.

3. $f_{max}(P^*, \underline{P}^*) = 0$
   Since P^* is a predecessor set, from the proof of Lemma 2 we know that $f_{max}(P^*, \underline{P}^*) \le 0$. If $f_{max}(P^*, \underline{P}^*) < 0$, it means that there is a pair of events $e_1 \in P^*$ and $e_2 \notin P^*$ such that $f_{max}(e_1, e_2) < 0$. This means that the residual capacity from e_1 to e_2 is positive and therefore there is an edge $e_1 \rightarrow e_2$ in the residual network. But this means that $e_2 \in P^*$, which is a contradiction.

Applying the properties of P^* to the relation in Lemma 1 we obtain $\Delta(P^*) = r_{max}(P^{*+}) - r_{max}(P^{*-}) + f_{max}(P^*, \underline{P}^*) = r_{max}(P^{*+}) \ge r_{max}(e^+) > 0$. □

It is now easy to find the predecessor set P_{max} with maximum positive resource level increment.

Theorem 2: *Consider all events $e^+_i \in R_H$ such that $r_{max}(e^+_i) > 0$ and consider the event set $P_{max} = \cup_{e+i}\ P^*_i$ where P^*_i is the set of events reachable from e^+_i in the residual network of f_{max}. Then P_{max} is a set of predecessors in R_H with maximum $\Delta(P_{max}) > 0$.*

Proof: Each of the P^*_i has the properties proved in Theorem 1. We show that P_{max} also has those properties.

1. Being the union of predecessor sets, P_{max} is also a predecessor set.
2. We know that $r_{max}(e^-) = 0$ for each $e^- \in P^*_i$. Therefore $r_{max}(P^-_{max}) = 0$.
3. From Lemma 2 we know that since $f(P^*_i, \underline{P}^*_i)=0$, there is no flow from events in $\underline{P}^*_i$ to events in P^*_i. Therefore there is also no flow from events in $\underline{P}^*_{max} = \cap_i \underline{P}^*_i$ to events in P^*_{max}. Hence, from Lemma 2 $f(P^*_{max}, \underline{P}^*_{max})=0$.

Therefore $\Delta(P_{max}) = r_{max}(P^+_{max}) > 0$.
Moreover, since by construction P_{max} contains all e^+_i with $r_{max}(e^+_i) > 0$. Therefore, for any other predecessor set P_X it is $r_{max}(P^+_X) \le r_{max}(P^+_{max})$. Hence, $\Delta(P_X) \le r_{max}(P^+_X) - r_{max}(P^-_X) \le r_{max}(P^+_X) \le r_{max}(P^+_{max}) = \Delta(P_{max})$. □

So far we have constructed P_{max} from a specific maximum flow for $F(R_H)$. However, it turns out that P_{max} is unique for all maximum flows of $F(R_H)$. Moreover P_{max} contains the minimum number of events among all predecessor sets with maximum positive resource level increment.

Theorem 3: *For any solution of the maximum flow problem for $F(R_H)$, P_{max} is the minimal predecessor set with maximum resource level increment $\Delta(P_{max})$.*

Proof: Consider the set $\{f_{max,i}\}$ with $i=1, \ldots, n$ of all n different maximum flows of $F(R_H)$. Since each corresponding $P_{max,i}$ is a maximum positive resource level increment sets, $\Delta(P_{max,i}) = \Delta_{max}$. Also, given two distinct maximum flows i and k, we have $P_{max,i} = P_{i \cap k} \cup P_{k-i}$ where $P_{i \cap k} = P_{max,i} \cap P_{max,k}$ and $P_{k-i} = P_{max,k} - P_{max,i}$. In the following we will indicate with $r_j(e)$ the residual $r_{max}(e)$ computed in flow $f_{max,j}$.
First we observe that for any two distinct i and k, $\Delta(P_{i \cap k}) = \Delta_{max}$. In fact, $P_{i \cap k}$ is a predecessor set and $r_i(P_{i \cap k}^-) = r_k(P_{i \cap k}^-) = 0$. Therefore $\Delta(P_{i \cap k}) = r_i(P_{i \cap k}^+) = r_k(P_{i \cap k}^+)$. We need now to show that $r_i(P_{i \cap k}^+) = r_i(P_{max,i}^+) = r_i(P_{max,k}^+)$. In fact, it must be $r_i(P_{k-i}^+)=0$ since $P_{max,i}$ must contain all e^+ events with $r_i(e^+) > 0$. Also, $\Delta_{max} = r_i(P_{max,i}^+) = \Delta(P_{max,k}) \le r_i(P_{max.k}^+) - r_i(P_{max,k}^-) \le r_i(P_{max,k}^+)$ which implies $r_i(P_{i-k}^+)=0$. But this means that the flow in $\Delta(P_{i \cap k})$ is self contained, i.e., there is no edge in the residual network of flow $f_{max,i}$ that exits $P_{i \cap k}$. Therefore, in this flow none of the events in P_{i-k} is reachable from an e^+ event and therefore $P_{i-k} = \varnothing$. With a symmetric argument we can see that $P_{k-i} = \varnothing$. Therefore for any i and k it must be $P_{max,i} = P_{max,k} = P_{max}$. The minimality of P_{max} derives from applying to $P_{max} - P^{\varnothing}$ the same argument used to demonstrate that P_{i-k} is empty, where $P^{\varnothing} \subseteq P_{max}$ is a predecessor graph with maximum positive resource level increment. □

Building Resource Envelopes

So far we know that the resource level at time $t \in H$ for a given schedule s is the sum of the weights of the events in C_H plus those of the events in some predecessor set P_X. It is not immediately obvious that the converse also applies, i.e., that given any predecessor set P_X one can determine a time $t_X \in H$, the *separation time,* and a schedule s_X, the *separation schedule,* such that all and only the events in $C_H \cup P_X$ are scheduled at or before time t_X. The reason this is not obvious is that events are still constrained by upper bound constraints, i.e., the metric links that are not included in **Pred(R).** Scheduling some event too early with respect to t_X may therefore force some event to occur before time t_X whether the event is a successor in **Pred(R)** or not. We will show that indeed we can find a separation time and schedule for *any* P_X and therefore also for P_{max}. For the latter we will show that t_X represents one of the times at which the resource level is maximum over **H** for any schedule. This will yield the resource envelope L_{max} if we reduce **H** to a time **t** and scan **t** over the horizon **T.**

Latest events

The first step is to identify the events in P_X that will be scheduled at time t_X. We say that **e** is a *latest event* of P_X if it is not a strict predecessor of any other event in P_X, i.e., for any $e_1 \in P_X$, $|e_1\ e| \ge 0$. There must be at least one latest event in P_X. If not, for every event $e_k \in P_X$, there would be an event $e_i \in P_X$ such that $|e_i\ e_k| < 0$. But this would mean that it would be possible to create a cycle $e_{i(1)} \rightarrow e_{i(2)} \rightarrow \ldots \rightarrow e_{i(n)}$ linked by links $|e_{i(k)}\ e_{i(k+1)}| < 0$, which is a contradiction to the hypothesis of temporal consistency of **R.** We will call $P_{X,late}$ the set of all latest events in P_X. Also, we define $P_{X,early} = P_X - P_{X,late}$.

The following properties hold for the temporal relations between events in $P_{X,\ late}$, $P_{X,\ early}$ and $\underline{P}_X$.

Property 1: *For any two events $e_1, e_2 \in P_{X,\ late}$, $|e_1 e_2| \ge 0$ and $|e_2\ e_1| \ge 0$.*

Property 2: *For any two events $e_1 \in \underline{P}_X$ and $e_2 \in P_{X,\ late}$, $|e_2\ e_1| > 0$.*
If not, e_1 would belong to P_X by the definition of P_X.

Property 3: *Any event $e_1 \in P_{X,early}$ is a strict predecessor of some $e_2 \in P_{X,\ late}$, i.e., $|e_2 e_1| < 0$.*

If not, consider any two events $\mathbf{e_1,e_3} \in P_{X,early}$. For any $\mathbf{e_2}$ it would be $|\mathbf{e_2e_1}|=0$ and $|\mathbf{e_2e_3}|=0$. Therefore, $0=|\mathbf{e_2e_3}| \leq |\mathbf{e_2e_1}| + |\mathbf{e_1e_3}| = |\mathbf{e_1e_3}|$, i.e., $|\mathbf{e_1e_3}| \geq 0$. Since this would be true for any pair of events in $\mathbf{P_{X,early}}$ and for all distances between any event in $\mathbf{P_{X,early}}$ and any event in $\mathbf{P_{X,late}}$, all events in $\mathbf{P_{X,early}}$ would be latest events, i.e., $\mathbf{P_{X,early}}=\varnothing$.

Separation Time for Latest Events

We now show how to construct the separation time $\mathbf{t_X}$ at which we will schedule all latest events.

Lemma 3: *There is a time interval* $[t_{X,min}, t_{X,max}]$ *that is in common among all time bounds* $[et(e), lt(e)]$ *with* $e \in P_{X,late}$ *and such that* $start(H) \leq t_{X,max}$.
Proof: First, we show that there must be a time value in common among all time bounds. If not, there would be two events $\mathbf{e_1, e_2} \in \mathbf{P_{X,late}}$ such that $\mathbf{et(e_1) > lt(e_2)}$. From the triangular inequality we also have that $|\mathbf{e_1e_2}| \leq - \mathbf{et(e_1)} + \mathbf{lt(e_2)} < 0$, which is inconsistent with Property 1. Now, assume $\mathbf{start(H)} > \mathbf{t_{X,\, max}}$. By the way the interval $[\mathbf{t_{X,\, min}}, \mathbf{t_{X,\, max}}]$ is constructed, there must exist an event $\mathbf{e} \in \mathbf{P_{X,\, late}}$ such that $\mathbf{lt(e)} = \mathbf{t_{X,\, max}}$. For this event it would be $\mathbf{lt(e)} < \mathbf{start(H)}$ that is a contradiction with **e** belonging to $\mathbf{R_H}$.□

We define $\mathbf{t_X}= \mathbf{max\ (start(H), t_{X,\, min})}$, with $\mathbf{t_X = start(H)}$ if $\mathbf{P_X}=\varnothing$, in which case $\mathbf{t_X = start(H)}$. We can then show that the time bound of each event in $\underline{\mathbf{P}}_\mathbf{X}$ indicates that each of them can be scheduled after $\mathbf{t_X}$.

Lemma 4: *For any event* $e \in \underline{P}_X$, $lt(e_1) > t_X$
Proof: By definition of $\mathbf{R_H}$ it must be $\mathbf{lt(e) > start(H)}$. So we only need to consider the case in which $\mathbf{t_X} = \mathbf{t_{X,\, min}} > \mathbf{start(H)}$. In this case there is at least one event $\mathbf{e_1} \in \mathbf{P_{X,\, late}}$ such that $\mathbf{et(e_1)} = \mathbf{t_{X,\, min}}$. For this event it is $|\mathbf{e_1\, e}| \leq - \mathbf{et(e_1)} + \mathbf{lt(e)}$. From Property 2 we know that $|\mathbf{e_1\ e}| > 0$. Therefore, $\mathbf{lt(e)} \geq \mathbf{et(e_1)} + |\mathbf{e_1\, e}| > \mathbf{et(e_1)} = \mathbf{t_{X,\, min}}$.□

Separation schedule for predecessors

We now show how to build a separation schedule $\mathbf{s_X}$ for $\mathbf{P_X}$ and $\mathbf{t_X}$, i.e., a schedule such that $\mathbf{s_X(e)} \leq \mathbf{t_X}$ for $\mathbf{e} \in \mathbf{C_H} \cup \mathbf{P_X}$ and $\mathbf{s_X(e)} > \mathbf{t_X}$ for $\mathbf{e} \in \underline{\mathbf{P}}_\mathbf{X} \cup \mathbf{O_X}$. Note that the following discussion holds also if $\underline{\mathbf{P}}_\mathbf{X}=\varnothing$.
We will do this with the following algorithm.

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

To show that $\mathbf{s_X}$ is a schedule we need to see that it is consistent with respect to **R**. We see that step 1 is consistent since: 1) $\mathbf{t_X}$ belongs to the intersection of all latest event time bounds; 2) since for any pair of latest events $|\mathbf{e_1e_2}| \geq 0$, scheduling one at $\mathbf{t_X}$ does not prevent any other latest events to be scheduled at time $\mathbf{t_X}$ as well. Step 3 above is also consistent since 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 $\mathbf{s_X}$. Note that we already know that it is satisfied for events in $\mathbf{P_{X,late}}$. By definition of $\mathbf{C_H}$ and $\mathbf{O_H}$, we also know that it is satisfied for events in these two sets. Therefore, we need to show that it is satisfied for $\mathbf{P_{X,early}}$ and $\underline{\mathbf{P}}_\mathbf{X}$.

a) $\mathbf{lt'(e)} \leq \mathbf{t_X}$ for all $\mathbf{e} \in \mathbf{P_{X,early}}$
   According to Property 3 we can pick an event $\mathbf{e_1} \in \mathbf{P_{X,\, late}}$ that $|\mathbf{e_1\ e}| < 0$. From the triangular inequality we have $\mathbf{lt'(e)} \leq \mathbf{lt'(e_1)} + |\mathbf{e_1\, e}| < \mathbf{lt'(e_1)} = \mathbf{t_X}$.
b) $\mathbf{lt'(e)} > \mathbf{t_X}$ for all $\mathbf{e} \in \underline{\mathbf{P}}_\mathbf{X}$.
   From Lemma 4 we know that before the re-propagation it was $\mathbf{lt(e)} > \mathbf{t_X}$. After the propagation, either $\mathbf{lt'(e)=lt(e)}$, in which case the condition is satisfied, or $\mathbf{lt'(e)}$ has changed with a propagation starting from some event $\mathbf{e_1} \in \mathbf{P_{X,\, late}}$. So it must be $\mathbf{lb'(e)} = \mathbf{t_X} + |\mathbf{e_1e}|$ and since from Property 2 $|\mathbf{e_1e}| > 0$, $\mathbf{lt'(e)} > \mathbf{t_X}$.

We can now compute $\mathbf{L_{max}}$ over the entire time horizon **T.** $\mathbf{P_{max}(R_H)}$ indicates that it is computed over $\mathbf{F(R_H)}$.

Theorem 4: *The maximum resource consumption 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 $\mathbf{t} \in \mathbf{H}$ the events in $\mathbf{R_H}$ that are scheduled before **t** are a predecessor set $\mathbf{P_X}$. For the resource level at time **t** it is always $\Delta\ (\mathbf{C_H}) + \Delta(\mathbf{P_X}) \leq \Delta\ (\mathbf{C_H}) + \Delta(\mathbf{P_{max}(R_H)})$, the latter being the resource level at the separation time for the separation schedule. This is true also if $\mathbf{P_{max}(R_H)}$ is empty.□

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 over time has the same complexity of 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 envelope. We now need to find an efficient algorithm.

Efficient Computation of Resource Envelopes

From Corollary 3, the naïve approach to compute a resource envelope would be to iterate over all possible $\mathbf{t} \in \mathbf{T}$. We can improve the running time by considering that we only need to compute $\mathbf{L_{max}}$ at times when either $\mathbf{C_t}$ or $\mathbf{R_t}$ changes. It is easy to see that this can only happen at $\mathbf{et(e)}$ or $\mathbf{lt(e)}$ for any $\mathbf{e} \in \mathbf{E(R)}$. Therefore we need to re-compute a maximum flow for a partial network in **R** only **2n** times, a substantial improvement over $|\mathbf{T}|$.
The complexity of some known flow algorithms is described in Table 1(Cormen, Leiserson, Rivest 1990). Note that the number of edges **E** is $\mathbf{O(V^x)}$ where V is the number of events and $\mathbf{1 \leq x \leq 2}$. Therefore the complexity of maximum flow algorithm is always $\mathbf{O(V^k\ lg^j(V))}$ with $\mathbf{1 \leq k \leq 5}$ and $\mathbf{j} \in \{\mathbf{0, 1}\}$. Let us now consider the worst case complexity of re-computing a flow at $\mathbf{et(e)/lt(e)}$. In the worst case, $\mathbf{C_t}$ will remain empty at all times and the size of each $\mathbf{R_t}$ will increase by **1** for each computation of the flow. Therefore the worst case complexity of this method is $\mathbf{O(\Sigma_{i=1,\ldots,V}O(i^k lg^j(i))=O(V^{k+1}lg^j(V))}$, a polynomial of higher order than maximum flow.

We can do better. Assume sorting all earliest and latest times in ascending order to yield a set **{t(1), t(2), …, t(2n)}.** Suppose now that when we compute the maximum flow for **$F(R_{t(i)})$** we make as much use as possible of the maximum flow for **$F(R_{t(i-1)})$.** In this case we can come up with an algorithm with the same worst-case complexity as computing the maximum flow on the entire network.

Incremental Change of Pending Events

Before we introduce the algorithm, let us consider the differences between **$R_{t(i)}$** and **$R_{t(i-1)}$ (1<i≤n)**. The first difference **$\delta(C_{t(i)}) = R_{t(i-1)} – R_{t(i)}$** is the sets of events **e** such that **t(i) = lt(e).** They must move from **$R_{t(i-1)}$** to **$C_{t(i)}$** at time **t(i).** The second difference, **$\delta(R_{t(i)}) = R_{t(i)} - R_{t(i-1)}$** are the events **e** such that **t(i) = et(e).** They must move from **$O_{t(i-1)}$** to **$R_{t(i)}$** at time **t(i).**

Figure 7 gives a complete picture of how all relevant event sets change at time **t(i).** In this picture **$E_{max}(t(i-1)) = C_{t(i-1)} \cup P_{max}(R_{t(i-1)})$** is the set of events needed to compute the resource envelope at time **t(i–1).** **$E_{max}(t(i)) = C_{t(i)} \cup P_{max}(R_{t(i)})$** is used to compute the resource envelope at time **t(i),** with **$C_{t(i)} = C_{t(i-1)} + \delta(Ct(i))$.** The difference **$E_{max}(t(i)) – E_{max}(t(i-1))$** can be separated into two disjoint sets **$\delta(C_{t(i)})– P_{max}(R_{t(i-1)})$** and **$P_{max}(R_{t(i)}) – P_{max}(R_{t(i-1)})$.** The goal of the efficient envelope algorithm is to identify the set **$P_{max}(R_{t(i)}) – P_{max}(R_{t(i-1)})$** with less effort than computing **$P_{max}(R_{t(i)})$** and **$P_{max}(R_{t(i-1)})$** with separate maximum flow computation and then differentiating them.

Ford-Fulkerson	$O(E \lvert f^* \rvert)$
Edmonds-Karp	$O(VE^2)$
Simple preflow-push	$O(V^2E)$
Preflow-push	$O(V^3)$
Goldberg-Tarjan	$O(VE\ lg(V^2/E))$

Table 1: Complexity of known maximum flow algorithms

Before proceeding notice that Figure 7 assumes that **$P_{max}(R_{t(i)}) \cap P_{max}(R_{t(i-1)}) \subseteq P_{max}(R_{t(i)})$.** As we will see this is indeed the case. The consequence of this is that as soon as **$P_{max}(R_{t(i-1)})$** has been determined and accounted for in the envelope calculation, the subnetwork of **$F(R_{t(i-1)})$** consisting of all events in **$P_{max}(R_{t(i-1)})$** and all incoming and outgoing edges can be deleted. This allows the computation of **$P_{max}(R_{t(i)})–P_{max}(R_{t(i-1)})$** directly from the maximum flow of **$F(R_{t(i)}-P_{max}(R_{t(i-1)}))$** which can save significant work.

The second efficiency improvement is computing the maximum flow of **$F(R_{t(i)}-P_{max}(R_{t(i-1)})$** by incrementally modifying the flow of **$F(R_{t(i-1)}-P_{max}(R_{t(i-2)}))$** during the deletion of events in **$\delta(C_{t(i)})$** and the addition of events in **$\delta(R_{t(i)})$** while maintaining the maximum flow property.

Incremental Modification of Maximum Flow

Let us focus on modifications of the flow network that preserve the maximum-flow. To do so we introduce the concept of a *prefix* and *postfix* of a resource increment flow network **F.** Consider a partition of events in the network in two event sets, **Post(F)** and **Pref(F).** We say that **Post(F)** is a postfix of **F** and **Pref(F)** is a postfix of **F** if for each pair of events **$e_1 \in$ Post(F)** and **$e_2 \in$ Pref(F), $|e_2\ e_1| > 0$.** It is immediate to see that for any flow of **F** it can only be **$f(e_2, e_1) \le 0$.** Therefore the residual network contains an edge **$e_2 \rightarrow e_1$** only if there is an edge **$e_1 \rightarrow e_2$** in the flow network and there is a positive **$f(e_1, e_2)$** passing through it.

We can see that **$\delta(C_{t(i)})$** is a prefix of **F(A)** where **A** is any subset of **$R_{t(i-1)}$** that contains **$\delta(C_{t(i)})$**. In fact, consider a pair of events **$e_2 \in \delta(C_{t(i)})$** and **$e_1 \in A - \delta(C_{t(i)})$.** From the definition of **$\delta(C_{t(i)})$** we have **$lt(e_2) = t$** and **$lt(e_1) \ge t+1$.** From the triangular inequality **$lt(e_1) \le lt(e_2) + |e_2\ e_1|$** we can deduce **$|e_2\ e_1| \ge lt(e_1) - lt(e_2) \ge t + 1– t = 1 > 0$.** A similar argument applies to demonstrate that **$\delta(R_{t(i)})$** is a postfix of **F(B)** where **B** is any subset **$R_{t(i)}$** that contains **$\delta(R_{t(i)})$.**

We now introduce two flow modification operations: *flow reduction* and *flow expansion.*

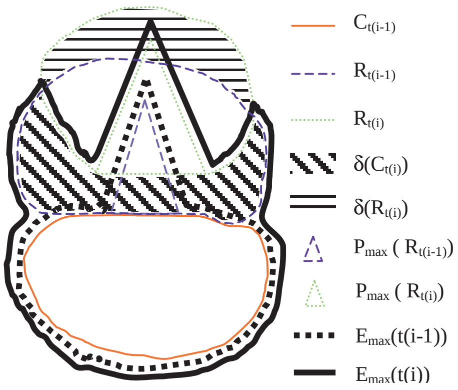


Figure 7 : Incremental change for set of pending events

Flow contraction: *Consider a network **F(A)**, a flow **f** for **F(A)** and a prefix of **A**, **Pref(A)**. Flow contraction consists of the following two steps:*

1) *while there is a flow-shifting path in the residual network connecting an e^- event in **Pref(A)** to an e^- event in **A-Pref(A)**, push flow along the path and update the residual network accordingly;*
2) *while there is a reducing path in the residual network connecting an e^- event in **Pref(A)** to an e^+ event in **A-Pref(A)**, push flow along the path and update the residual network accordingly.*

Lemma 5: *If the flow f_{max} is maximum for **F(A)**, flow contraction produces a maximum flow for **F(A-Pref(A)).***

Proof: If **f_{max}** is maximum, the flow **f’** produced at the end of step 1 is still a maximum flow for **F(A).** This because flow shifting does not affect **$f(\{s\}, A^+)$** since no flow is pushed back through any edge **$f(s, e^+)$**. At the end of step 2 we will have a flow **f”.** Note however that any modification of the residual network in step 2 can only eliminate existing edges and therefore eliminate paths**.** Since **f’** maximality implies that there are no augmenting paths in it going to events **$e^- \in$ A-Pref(A)**, there will be no such augmenting paths in **f”** either. Therefore **f”** will be maximum in **F(A- Pref(A)).□**

Note that in achieving the maximum flow for **F(A-Pref(A))** it is always better to us flow-shifting paths before reducing paths, since if flow needs to be moved from

Pref(A) to **A-Pref(A)** to achieve optimality, a flow-shifting path is always shorter than the concatenation of a reducing and an augmenting path.

Flow expansion: *Consider a network **F(A),** a postfix of **A, Post(A)),** a flow **f** for **F(A-Post(A)).** Flow expansion consists of the following steps:*

- *while there is an augmenting path in the residual network of **F(A)** connecting an e^+ event in **Post(A)** to an e^- event in **A,** push flow along the path and update the residual network accordingly.*

It is clear that flow expansion produces a maximum flow f_{max} for **F(A).** Also, if the starting flow for **F(A-Pref(A))** is a maximum flow, flow augmentation will minimize work. This is important in our application of maximum flow to a sequence of **n** $F(R_{t(k)})$ problems since re-doing unnecessary work may negatively impact asymptotic complexity.

Flow Network Reduction

Now we will show that it is always safe to eliminate any events in $P_{max}(R_{t(i-1)})$ from consideration once their impact on $L_{max}(t(i-1))$ has been recorded. Consider the set $\delta(C_{t(i)})\cap P_{max}(R_{t(i-1)})$. The effect of these events is already included in $L_{max}(t(i-1))$ and therefore their contribution does not need to be included in $L_{max}(t(i))$. Let us now consider the effect that flow reduction applied to these events has on the maximum flow of $F(R_{t(i-1)} \cap R_{t(i)})$. From the property of the predecessor set with maximum resource level increment, we know that $f(e_1,e_2)=0$ and $f(e_2,e_1)=0$ with $e_1\in \underline{P}_{max}(R_{t(i-1)})$ and $e_2\in P_{max}(R_{t(i-1)})$. Therefore, the residual network for the maximum flow of $F(R_{t(i-1)})$ does not have any edges $e_2\rightarrow e_1$. Also, since all $e^- \in P_{max}(R_{t(i-1)})$ are saturated ($r_{max}(e^-)=0$), there cannot be any edge $e_2\rightarrow t$ in the residual network. Therefore, there cannot be any flow shifting and the only way to push flow back from an $e^-\in P_{max}(R_{t(i-1)})$ is to do so through an $e^+\in P_{max}(R_{t(i-1)})$. Therefore, flow changes during flow contraction due to events in $P_{max}(R_{t(i-1)})$ do not affect the rest of the network.

Consider now $P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$. We know that after contraction the flow is maximum for $F(R_{t(i-1)}\cap R_{t(i)})$. Therefore all $e^-\in P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$ are still saturated. Also, there will be at least one $e^+\in P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$ with $r_{max}(e^+)>0$. If we now add $\delta(R_{t(i)})$ and apply flow expansion, we know that throughout the process there will be no augmenting path exiting from $e^-\in P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$. At the end of the process, $P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$ will still be isolated and therefore $\Delta(P_{max}(R_{t(i-1)})-\delta(C_{t(i)}))>0$. Therefore $P_{max}(R_{t(i)})\supseteq P_{max}(R_{t(i-1)})-\delta(C_{t(i)})$. Therefore, $\Delta(P_{max}(R_{t(I-1)}))$ will be part of both $L_{max}(t(i-1))$ and $L_{max}(t(i))$. Moreover, the presence of $P_{max}(R_{t(i-1)})$ has no effect on the flow modification operations and therefore all events in $P_{max}(R_{t(i-1)})$ together with their incoming and outgoing edges can be safely eliminated from any flow network considered by the algorithm after $F(R_{t(i-1)})$.

Figure 8 shows the pseudocode of the algorithm. Here we assume that after executing flow contraction , the function **Flow_Contraction** function also deletes from the network **F** the portion pertaining to the events in $p.E_{close}$. Similarly we expect **Flow_Expansion** to add to **F** the portion pertaining to E_{pend}. The function **Network_Reduction** deletes from the network **F** the portion pertaining to P_{max}. Finally, **Extract_P_max** finds the P_{max} in the maximum flow of **F** by collecting the events that are reachable in the residual network of **F** from each e^+ with $r_{max}(e^+)>0$.

Complexity Analysis

Now we can show that the complexity of **Resource_Envelope** is asymptotically the same as running a flow algorithm over the flow network **F(R)** for the entire cR-STN. We demonstrate this for the Ford-Fulkerson method. This is the simplest maximum flow method and tends not to perform well when the internal “pipes” of the flow network have bottlenecks (Cormen, Leiserson, Rivest 1990). However, this is not true for **F(R)** in which the capacity of all internal pipes is +∞. Therefore in our case the Ford-Fulkerson method should perform well. The complexity of Ford-Fulkerson is $O(E\ |f_{max}|)$, where **E** is the number of edges in the flow network and $|f_{max}|$ is the amount of the maximum flow pushed through the network.

```
Resource_Envelope (R, Pred(R))
{   $L_{old}$ := 0;          /* envelope level at previous iteration time */
     $P_{max}$ := ∅          /* temporary variable to hold $P_{max}$ */
    Envelope := ∅;          /* envelope profile, a list of pairs e=<t, L> where t is the
                               time at which the envelope reaches level L. The
                               envelope stays constant at L until the time of the
                               following entry in the list     */
    F := ∅;                 /* auxiliary flow graph for $F(R_{t(i)}-P_{max}(R_{t(i-1)})$*/
    E := {sorted list in ascending order of t of triples <$E_{close}$, $E_{pend}$, t> where $E_{close}$
            is the set of events e in Pred(R) such that lt(e) = t and $E_{pend}$ is the list
            of events e in Pred(R) such that et(e) = t};
    t := 0
    while (E is not empty)
    {   $E_{cur}$   := ∅;
        p := pop(E);
        t := p.t;
        F := Flow_Contraction(F, p.$E_{close}$);
        F := Flow_Expansion(F, p.$E_{pend}$, Pred(R));
        $P_{max}$ := Extract_P_max (F);
        F := Network_Reduction(F, $P_{max}$);
        $L_{new}$ := $L_{old}$ + $\Delta(E_{close})$ + $\Delta(P_{max})$;
        Envelope := append (<t, $L_{new}$>, Envelope);
        $L_{old}$ := $L_{new}$;
    }
}
```

Figure 8: Maximum resource envelope algorithm .

Let us consider separately the total cost of the three flow propagations in **Flow_Expansion** and **Flow_Contraction**: the one across augmenting paths, the one across flow-shifting paths and the one across reducing paths.

Consider the propagation across augmenting paths. Consider the total flow that we will be able to push from the source throughout these propagations. Since when we push flow from e^+ events in $\delta(R_{t(i)})$, some events e^- may have become unavailable (through deletion) or saturated (due to flow shifting). Therefore the total flow we can push in **Flow_Contraction** is no greater than $|f_{max}|$ in **F(R).** Moreover, the set of edges over which the search for augmenting paths is conducted is always not greater in each invocation of **Flow_Expansion** than in the application of maximum flow to the entire **$F(R)$.** Therefore, the total cost of **Flow_Expansion** is **$O(E\ |f_{max}|)$**

Consider now the propagation across flow-shifting paths in **Flow_Contraction.** Consider an auxiliary flow graph **$Shift(R)$** built as follow: for each **$\delta(C_{t(i)})$** consider all edges in the flow-shifting paths before the application of **Flow_Contraction(F, $\delta(C_{t(i)})$)** and for each pair of events **e_1** and **e_2** (including **σ** and **τ**) add the link with maximum capacity in some of the flow-shifting paths across all invocations of **Flow_Contraction**. The total number of edges in **$Shift(R)$** is bounded by **$2\ E + 2\ V$,** since this is the maximum number of edges in a residual network. Considering now pushing flow from an **e^-** in **$\delta(C_{t(i)})$,** we notice that the pipes that are available in **$Shift(R)$** are no smaller than those available in the flow-shifting paths during the execution of **Flow_Contraction.** Moreover, the lengths of the paths is no shorter in **$Shift(R)$** than in each invocation of **Flow_Contraction** since events are not deleted in **$Shift(R)$.** Finally, the total flow that can be shifted is no greater than **$|f_{max}|$**, since this is an upper bound of the maximum amount of flow that reaches **τ** from some **e^-.** From this argument we can see that the total amount of work done during the flow-shifting step **Flow_Contraction** is bounded by **$O(E\ |f_{max}|)$.** A similar argument applies to the flow reduction step in **Flow_Contraction** with its complexity again bound by **$O(E\ |f_{max}|)$.** Therefore the total cost of **Flow_Contraction** and **Flow_Expansion** in **Resource_Envelope** is bounded by **$3\ O(E\ |f_{max}|)$**.

Considering the other steps in **Resource_Envelope,** the sorting step to initialize **E** is **$O(V\ lgV)$,** and the total cost of **Extract_P_max** and of incrementally constructing and deleting the flow network**,** is 3 **$O(E)$.** Under the reasonable assumption that the asymptotic cost of flow computation dominates **$O(V\ lgV)$,** the total cost of **Resource_Envelope** is **$O(E\ |f_{max}|)$,** i.e., it has the same asymptotic complexity than running the flow algorithm once over the entire **$F(R)$.**

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 looser bound computations. Future work includes testing the practical effectiveness of resource envelopes in scheduling search for problems with multi-capacity resources. This includes both direct use as a backtracking and termination criterion in a constrained based scheduling algorithm for multi-capacity resources and the additional development of effective variable and value ordering heuristics based on resource envelopes.

References

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.

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.

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

T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. Cambridge, MA, 1990.

R. Dechter, I. Meiri, J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61-95, May 1991.

P. Laborie, Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results, *Proceedings of ECP 2001*, Toledo, Spain, 2001.

P. Morris, N. Muscettola, T. Vidal. Dynamic Control of Plans with Temporal Uncertainty, in *Proceedings of IJCAI 2001*, Seattle, WA, 2001

N. Muscettola. On the Utility of Bottleneck Reasoning for Scheduling. in *Proceedings of AAAI 1994*, Seattle, WA, 1994.

W.P.M. Nuijten. Time and Resource Constrained Scheduling: a Constraint Satisfaction Approach. PhD Thesis, Eindhoven University of Technology, 1994.

N. Sadeh. Look-ahead techniques for micro-opportunistic job-shop scheduling. PhD Thesis, Carnegie Mellon University, CMU-CS-91-102, 1991.

I. Tsamardinos, N. Muscettola, P. Morris. Fast Transformation of Temporal Plans for Efficient Execution. in *Proceedings of AAAI 1998*, Madison, WI, 1998.