Incremental Maximum Flows for Fast Envelope Computation

Nicola Muscettola

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

Abstract

Resource envelopes provide the tightest exact bounds on the resource consumption and production caused by all possible instantiations of a temporally flexible plan. We present a new algorithm that computes an envelope in **O(Maxflow(n, m, U))** where **n, m** and **U** measure the size of the flexible plan. This is an **O(n)** improvement on the best envelope algorithm known so far and makes envelopes more amenable to practical use in scheduling algorithms. The reduction in complexity depends on the fact that when the algorithm computes the constant segment **i** of the envelope it makes full reuse of the maximum flow that was computed in order to obtain segment **i-1**.

Resource Envelopes

The execution of plans greatly benefits from temporal flexibility. Fixed-time plans are brittle and may require extensive replanning due to execution uncertainty. Moreover, when plans must deal with uncontrollable exogenous events (Morris et al., 2001) temporal flexibility cannot be avoided. However, effective algorithms to build temporally flexible plans are rare, especially when activities produce or consume variable amounts of resource capacity. A major obstacle is the difficulty of assessing the resource needs across all possible plan executions. Methods are available to compute resource consumption bounds (Laborie, 2001; Muscettola, 2002). In particular, (Muscettola, 2002) proposes a polynomial algorithm to compute a *resource envelope*, the tightest of these bounds. By being the tightest possible, resource envelopes can potentially save an exponential amount of search (through early backtracking and solution detection) when compared to using looser bounds. Also, methods that compute resource envelopes identify maximally matched sets of resource consumer/producers that balance each other for any plan execution. This and other structural information could be crucial in minimizing the search space and suggesting effective scheduling heuristics, potentially enabling new classes of highly efficient schedulers.

However, preliminary studies on schedulers using envelopes appear not to show advantages with respect to more traditional heuristic methods based on fixed-time resource profiles (Pollicella et al., 2003). When compared to traditional fixed-time profiling methods, it is critical to balance the increased computation cost with the extraction of more structural problem information from the envelope than backtrack/termination tests and maximum resource contention intervals. Making the trade-off advantageous requires two complementary approaches. The first reduces the cost of computing an envelope; the second devises new envelope analysis methods to extract useful heuristics.

In this paper we address the problem of cost reduction. The fastest known resource envelope algorithm (Muscettola, 2002) computes all piecewise-constant segments of the envelope through as many as **2n** stages, where **n** is the number of events in the flexible plan. Each stage computes a maximum flow using some maximum flow algorithm. The worst case complexity is **O(n Maxflow(n,m,U))** where **m** is the number of temporal constraints between activities in the plan, **U** is the maximum level of resource production or consumption at some activity, and **Maxflow(n, m, U)** is the asymptotic cost of the maximum flow algorithm.

This staged method, however, can be significantly improved since at each stage it recomputes the needed maximum flow completely from scratch. This suggests using an incremental flow method. Starting from the maximum flow at one stage, this method computes the maximum flow at the next stage by minimally reducing flow when deleting nodes and edges, and by minimally increasing flow when adding new nodes and edges (Kumar, 2003). However, without appropriately ordering flow reductions and increases, the asymptotic complexity may not improve (at it appears to be the case in (Kumar, 2003)).

In this paper we introduce an incremental method that provably computes an envelope in **O(Maxflow(n, m, U))** for a large class of maximum flow algorithms**.** This reduction of complexity is significant. Experimental analysis has shown that the practical cost of maximum flow is usually as low as $\mathbf{O(n^{1.5})}$ (Ahuja et al., 1993). This compares well with **O(n log n),** the cost of building resource profiles for fixed time schedules. This paper is organized as follows. We first give a succinct introduction to the resource envelope problem and the staged envelope algorithm in (Muscettola, 2002). Next we present the new incremental algorithm identifying all sources of performance improvements. We then prove the complexity result and conclude by discussing future work.

Staged Computation of Envelopes

In this section we introduce the essential information on the envelope problem and the staged algorithm that solves it. For a complete discussion, see (Muscettola, 2002).

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 $\mathbf{e_{4s}}$). Two additional events $\mathbf{T_s}$, and $\mathbf{T_e}$ define a time horizon within which all events occur.

Time origin, events and links constitute a Simple Temporal Network. To describe resource production and consumption each event has also an *allocation value* (e.g., $\mathbf{r_{31}}$ for event $\mathbf{e_{3s}}$), a numeric weight that represents the amount of resource allocated when the event occurs. We will assume that all allocations refer to a single, multi-capacity resource. The extension to multiple resources is straightforward. If the allocation is negative an event $\mathbf{e^-}$ is a *consumer*, if it is positive $\mathbf{e^+}$ is a *producer*. We assume that the temporal constraints are consistent which means that for any pair of events the shortest path $\mathbf{|e_1e_2|}$ from $\mathbf{e_1}$ to $\mathbf{e_2}$ is well defined. Each event **e** can occur within its time bound, between the earliest time $\mathbf{et(e) = -|eT_s|}$ and the latest time $\mathbf{lt(e) = |T_se|}$. The triangular inequality $\mathbf{|e_1e_3| \leq |e_1e_2| + |e_2e_3|}$ holds for any three events $\mathbf{e_1}$, $\mathbf{e_2}$ and $\mathbf{e_3}$.

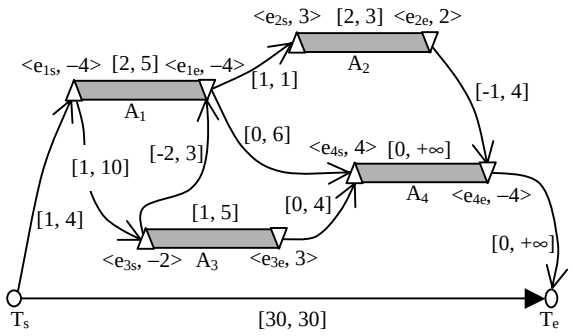

Figure 1: An activity network with resource allocations

The *anti-precedence graph*, **Aprec,** is a graph containing a path between any two events $\mathbf{e_1}$ and $\mathbf{e_2}$ such that $\mathbf{|e_1\, e_2| \leq 0.}$ Figure 2 depicts an anti-precedence graph of the network in Figure 1 with each event labeled with its time bound and resource allocation.

We can now formally define a resource envelope. For any subset of events **A,** the *resource level increment* of **A** is $\mathbf{\Delta(A) = 0}$ if $\mathbf{A = \emptyset}$, and $\mathbf{\Delta(A) = \Sigma_{e \in A}\, c(e)}$ if $\mathbf{A \neq \emptyset}$. If **S** is the set of all possible consistent time instantiations for all events and **t** is a time within the time horizon, the resource level at time **t** for a specific time instantiation $\mathbf{s \in S}$ is $\mathbf{L_s(t) = \Delta(E_s(t))}$. Here $\mathbf{E_s(t)}$ is the set of events **e** which occur at or before **t** in **s**. The *maximum resource envelope* is $\mathbf{L_{max}(t) = max_{s \in S}\, L_s(t)}$ and the *minimum resource envelope* is $\mathbf{L_{min}(t) = min_{s \in S}\, L_s(t)}$. Since $\mathbf{L_{min}}$ can be computed with obvious term substitution on the method that computes $\mathbf{L_{max}}$, we only focus on $\mathbf{L_{max}}$.

To compute the resource envelope at time **t** we partition all events into three sets depending on the position of their time bound relative to **t**: 1) the *closed events* $\mathbf{C_t}$ that must occur before or at **t,** i.e., such that that $\mathbf{lt(e) \leq t;}$ 2) the *pending events* $\mathbf{R_t}$ that can occur before, at or after **t,** i.e., such that $\mathbf{(e) \leq t < lt(e)}$; and 3) the *open events* $\mathbf{O_t}$ that must occur strictly after **t**, i.e., such that $\mathbf{et(e) > t.}$

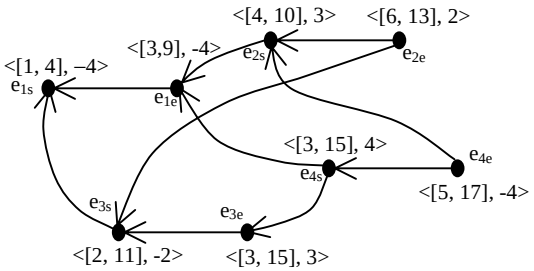

Figure 2: A resource increment flow network

Any resource level increment $\mathbf{L_s(t)}$ will always include the contribution of all events in $\mathbf{C_t}$ and none of those in $\mathbf{O_t}$ but can include only some subset of events in $\mathbf{R_t}$, i.e., only those that are scheduled before **t** in **s**. It is possible to show that this subset must be a *predecessor set* $\mathbf{P \subseteq R_t}$ such that if $\mathbf{e \in P}$ and **e'** follows **e** in **Aprec,** then $\mathbf{e' \in P.}$ We call $\mathbf{P_{max}(R_t)}$ the (possibly empty) predecessor set with maximum non-negative resource level increment.

The fundamental result reported in (Muscettola, 2002) is that $\mathbf{L_{max}(t)}$ can be determined from the following equation.

Equation 1: $\mathbf{L_{max}(t) = \Delta(C_t) + \Delta(P_{max}(R_t))}$

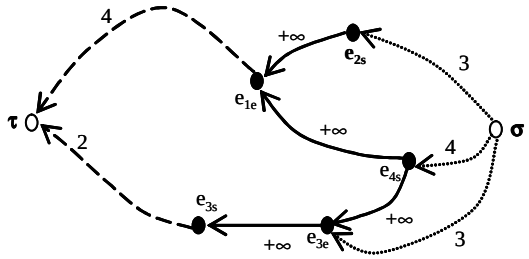

Figure 3: Anti-precedence graph with time/resource usage

The cost of computing an envelope depends on the cost of computing $\mathbf{P_{max}(R_t)}$. We can compute $\mathbf{P_{max}(R_t)}$ by solving a maximum flow problem on an auxiliary flow network $\mathbf{\mathit{F}\,(R_t)}$, the *resource increment flow network* for $\mathbf{R_t}$.

The formal definition of a resource increment flow network can be found in (Muscettola, 2002). As an example, Figure 2 gives $\mathbf{\mathit{F}\,(R_4)}$ for the activity network in Figure 1. The network has a node for each event in $\mathbf{R_4}$, an infinite capacity flow edge between two events for each edge in **Aprec** (see Figure 2), an edge from the source $\mathbf{\sigma}$ to a producer with capacity equal to the producer's allocation, and an edge from a consumer to the sink $\mathbf{\tau}$ with capacity equal to the opposite of the consumer's allocation.

A complete discussion of maximum flow algorithms can be found in (Cormen, Leiserson and Rivest, 1990). Here we only highlight a few concepts that we will use in the following. A flow is a function $\mathbf{f(e_1, e_2)}$ of pair of events in $\mathbf{\mathit{F}\,(R_t)}$ that is skew-symmetric, i.e., $\mathbf{f(e_2, e_1) = -\, f(e_1, e_2),}$ has

a value no greater than the capacity of edge $\mathbf{e_1 \rightarrow e_2}$ (assuming capacity zero if the edge is not in $\boldsymbol{F}$ $\mathbf{(R_t))}$, and is balanced, i.e., the sum of all flows entering an event must be zero. A pre-flow is a function defined similarly but that relaxes the balance constraint by allowing the sum of preflows entering a node to be positive. The total network flow is defined as $\Sigma_{e \in Rt}$ $\mathbf{f(\sigma, e)} = \Sigma_{e \in Rt}$ $\mathbf{f(e, \tau)}$. 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*, a graph with an edge for each pair of nodes in $\boldsymbol{F}$ $\mathbf{(R_t)}$ with positive *residual capacity*, i.e., the difference between edge capacity and flow. Each residual network edge has capacity equal to the residual capacity. An *augmenting path* is a path connecting $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$ in the residual network. The existence of an augmenting path indicates that additional flow can be pushed from $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$. Alternatively, the lack of an augmenting path indicates that a flow is maximum.

We can compute $\mathbf{P_{max}(R_t)}$ according to the next theorem.

Theorem 2: (Muscettola 2002) $\boldsymbol{P_{max}(R_t)}$ *is the (possibly empty) set of events that are reachable from the source* $\boldsymbol{\sigma}$ *in the residual network of some* $\boldsymbol{f_{max}}$ *of* $\boldsymbol{F}$ $\boldsymbol{(R_t)}$.

From Equation 1 and Theorem 2 (Muscettola, 2002) derives a staged envelope algorithm as follows. Consider the **2n** times $\mathbf{t_i}$ corresponding to the earliest and latest times for all events. Since the envelope level can only change at one of these times, the algorithm computes a different level for each of them. At a particular $\mathbf{t_i}$ the algorithm determines its closed and pending event sets $\mathbf{C_i}$ and $\mathbf{R_i}$, builds $\boldsymbol{F}$ $\boldsymbol{(R_i)}$, solves a maximum flow over it, determines $\mathbf{P_{max}(R_i)}$ according to Theorem 2, and computes $\mathbf{L_{max}(t_i)}$ according to Equation 1. It is easy to see that the worst-case time complexity of this algorithm is **O(n Maxflow(n, m, U)).**

Incremental Computation of Envelopes

In the previously described staged envelope algorithm flows are recomputed from scratch for each $\boldsymbol{F}$ $\mathbf{(R_i)}$. Assuming that the times t_i are sorted in increasing order, in To reduce cost, we can try and reuse as much as possible of the maximum flow computation performed on $\boldsymbol{F}$ $\mathbf{(R_{i-1})}$. At time $\mathbf{t_i}$ the set of pending events can undergo two modifications. First, the events $\mathbf{\delta C_i = R_{i-1} - R_i}$ move from $\mathbf{R_{i-1}}$ to $\mathbf{C_i}$. These are events **e** such that $\mathbf{t_i = lt(e)}$. Second, the events $\mathbf{\delta R_i = R_i - R_{i-1}}$ move from $\mathbf{O_{i-1}}$ to $\mathbf{R_i}$. These are the events **e** such that $\mathbf{t_i = et(e)}$. For example, consider the activity network in Figure 1 and the process through which $\mathbf{R_3}$ is transformed into $\mathbf{R_4}$. This is described in Figure 4 where the grayed part of the network is deleted and the emphasized part of the network is added at time 4. In particular, we have $\mathbf{\delta C_4 = \{e_{1s}\}}$ and $\mathbf{\delta R_4 = \{e_{2s}\}}$. For completeness, we note that $\boldsymbol{F}$ $\mathbf{(\delta C_4)}$ consists of node $\mathbf{e_{1s}}$ and edge $\mathbf{e_{1s} \rightarrow \tau}$ while $\boldsymbol{F}$ $\mathbf{(\delta R_4)}$ consists of node $\mathbf{e_{2s}}$ and edge $\mathbf{\sigma \rightarrow e_{2s}}$. All other added and deleted edges are connectives between $\boldsymbol{F}$ $\mathbf{(R_4 - \delta C_4)}$ and $\boldsymbol{F}$ $\mathbf{(\delta C_4)}$ (edges $\mathbf{e_{1e} \rightarrow e_{1s}}$ and $\mathbf{e_{3s} \rightarrow e_{1s}}$) and between $\boldsymbol{F}$ $\mathbf{(\delta R_4)}$ and $\boldsymbol{F}$ $\mathbf{(R_4 - \delta C_4)}$ ($\mathbf{e_{2s} \rightarrow e_{1e}}$).

The sets $\mathbf{\delta C_i}$ and $\mathbf{\delta R_i}$ satisfy the following fundamental properties.

Lemma 3: $\boldsymbol{\delta C_i}$ *is a predecessor set contained in* $\boldsymbol{R_i}$. $\boldsymbol{\delta R_i}$ *is the complement of predecessor set* $\boldsymbol{R_{i-1}}$ *in* $\boldsymbol{R_i}$.

Proof: We only give the proof for $\mathbf{\delta C_i}$ since the one for $\mathbf{\delta R_i}$ is analogous. Consider a pair of events $\mathbf{e_1 \in \delta(C_i)}$ and $\mathbf{e_2 \in R_{i-1} - \delta C_i}$. From the definition of $\mathbf{\delta C_i}$ we have $\mathbf{lt(e_1) = t_i}$ and $\mathbf{lt(e_2) \geq t_i + 1}$. From the triangular inequality $\mathbf{lt(e_2) \leq lt(e_1) + |e_1\, e_2|}$ we deduce $\mathbf{|e_1\, e_2| \geq lt(e_2) - lt(e_1) \geq t_i + 1 - t_i = 1 > 0}$.□

Lemma 3 determines what flow edges are eliminated when $\mathbf{\delta C_i}$ is deleted and what are added when $\boldsymbol{\delta R_i}$ is added. In particular, we can only delete edges that enter events in $\mathbf{\delta C_i}$ or go from $\mathbf{\delta C_i}$ to $\boldsymbol{\tau}$. Similarly, we can only add edges that exit events in $\boldsymbol{\delta R_i}$ or go from $\boldsymbol{\sigma}$ to $\boldsymbol{\delta R_i}$. Unlike previous proposals for incremental envelope calculation (Kumar, 2003), our methods relies on events and edges exiting and entering the current flow network in a well defined order. This is the primary key to reducing complexity in our incremental envelope algorithm.

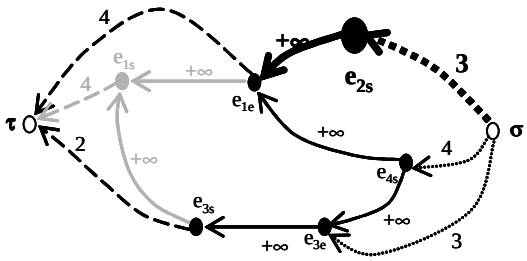

Figure 4: Incremental modification of a resource flow network

Directly related to Lemma 3 is the possibility of computing the maximum flow of $\boldsymbol{F}$ $\mathbf{(R_i)}$ by incrementally modifying the flow of $\boldsymbol{F}$ $\mathbf{(R_{i-1})}$, reusing both flow values and intermediate data structures across successive invocations of a maximum flow algorithm. We prove that our flow modification operators guarantee the optimality at each intermediate flow. Maintaining maximality of intermediate flows and reusing data structures across flows are keys to reduce complexity for different kinds of maximum flow algorithms.

A final factor is minimizing the size of each intermediate flow network. We will show that as soon as the weight of an intermediate $\mathbf{P_{max}}$ is used in the envelope calculation, $\boldsymbol{F}$ $\mathbf{(P_{max})}$ and all of its connecting edges can be safely eliminated from further consideration. This reduces flow network size and additionally contributes to cost reduction.

Flow Modification Networks

We now define the operators needed for the incremental envelope algorithm. The philosophy of each operator is similar to that used by the flow augmentation method in

maximum flow theory. However, we use this method more generally not only to augment flow but also to shift flow around the network and to reduce flow. The general idea is the following. Given a flow network F and one of its maximum flows $\mathbf{f}$, an operator first defines an auxiliary flow transformation network F_T, then finds one of its maximum flows $\mathbf{f_T}$, and finally produces a flow $\mathbf{f_{new} = f + f_T}$. Each F_T consists of selected edges in the residual network of F for $\mathbf{f}$. Since the properties of flows are preserved when adding a flow of a residual network to the flow that originated the residual network, $\mathbf{f_{new}}$ is also a flow for network F.

Consider now the resource increment flow network $F(\mathbf{R_{i-1}})$ at stage **i-1** and assume that the set of new closed events at stage **i** $\mathbf{\delta C_i}$ is not empty. At stage **i** all events in $\mathbf{\delta C_i}$ and all of its incoming and outgoing edges will be deleted. This also means that any flow that at the end of stage **i-1** enters $\mathbf{\delta C_i}$ will necessarily have to be zeroed, i.e., pushed back into $F(\mathbf{R_{i-1}})$. The value of this flow is the sum of the residual capacities of all edges $\mathbf{e_1 \rightarrow e_2}$ where $\mathbf{e_1 \in \delta C_i}$ and $\mathbf{e_2 \in R_{i-1} - \delta C_i}$. Once pushed back, this flow can follow two routes. The first reaches $\boldsymbol{\tau}$ to fill up some non-saturated exiting edges of $F(\mathbf{R_{i-1} - \delta C_i})$. The second reverses all the way to $\boldsymbol{\sigma}$ because it cannot find any way to exit $F(\mathbf{R_{i-1} - \delta C_i})$ through its sink. We call this flow push-back operation a *flow contraction*. The first flow route corresponds to a *flow shift* and the second one is a *flow reduction*. For example, consider the network in Figure 4. Assume that at **t=3** it is $\mathbf{f_{max}(e_{1s}, \tau) = 4}$, $\mathbf{f_{max}(e_{1e}, \tau) = 1}$ and $\mathbf{f_{max}(e_{3e}, \tau) = 2}$. At t=3 the elimination of $\mathbf{e_{1s}}$ requires pushing back 4 units of flow. However, note that three units can still reach $\boldsymbol{\tau}$ by being shifted to $\mathbf{e_{1e} \rightarrow \tau}$. Only one unit of flow needs to be pushed back to $\boldsymbol{\sigma}$. If we did not shift (as in (Kumar, 2003)), three additional units of flow would have to be pushed again from $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$ to ensure flow maximality.

Assume now that at stage **i** there is also a non-empty set $\mathbf{\delta R_i}$ of new pending events. Augmenting $F(\mathbf{R_{i-1} - \delta C_i})$ with the part of the resource increment flow network pertaining to $\mathbf{\delta R_i}$ yields $F(\mathbf{R_i})$. Assume now that $F(\mathbf{R_{i-1} - \delta C_i})$ is traversed by the flow resulting from flow contraction. Even if this flow is maximum for $F(\mathbf{R_{i-1} - \delta C_i})$, it may not be maximum for $F(\mathbf{R_i})$ since additional flow could be pushed through edges $\mathbf{\sigma \rightarrow e}$ with $\mathbf{e \in \delta R_i}$. We call this flow push-forward operation a *flow expansion*. If at every stage of flow contraction and flow expansion we guarantee flow maximality, we will obtain a maximum flow for $F(\mathbf{R_{t(i)}})$ that moves a minimal amount of flow.

Flow Contraction

Let us call $\mathbf{f_{max,i-1}}$ the maximum flow for $F(\mathbf{R_{i-1}})$. In our discussion we ignore the structure of the flow sub-network for $\mathbf{\delta C_i}$ by using an auxiliary flow network $\underline{F}_{\mathbf{i-1}}$ that redirects all flow entering $\mathbf{\delta C_i}$ into the sink $\boldsymbol{\tau}$. Formally, to obtain $\underline{F}_{\mathbf{i-1}}$ we first delete from $F(\mathbf{R_i})$ all events in $\mathbf{\delta C_i}$, together with all their incoming and outgoing flow edges. We then add an *auxiliary edge* $\mathbf{e_1 \rightarrow \tau}$ for each set of component edges $\mathbf{e_1 \rightarrow e_2}$ in $\mathbf{F(R_{i-1})}$ such that $\mathbf{e_1 \in R_{i-1} - \delta C_i}$ and $\mathbf{e_2 \in \delta C_i}$. The capacity of the auxiliary edge $\mathbf{e_1 \rightarrow \tau}$ is the sum of all component edge flows $\mathbf{f_{max,i-1}(e_1, e_2)}$. We call $\underline{\mathbf{f}}_{\mathbf{max,,i-1}}$ a function over the edges of $\underline{F}_{\mathbf{i-1}}$ where $\underline{\mathbf{f}}_{\mathbf{max,i-1}}\mathbf{(e_1,e_2)}$ is equal to $\mathbf{f_{max,i-1}(e_1,e_2)}$ if $\mathbf{e_1 \rightarrow e_2}$ also belongs to $F(\mathbf{R_{i-1}})$, and $\underline{\mathbf{f}}_{\mathbf{max,i-1}}\mathbf{(e_1,\tau)}$ is equal to the edge's capacity if $\mathbf{e_1 \rightarrow \tau}$ is one of the auxiliary flow edges. It is easy to see that $\underline{\mathbf{f}}_{\mathbf{max,i-1}}$ is a maximum flow for $\underline{\mathbf{F}}_{\mathbf{i-1}}$. We call $\underline{\mathbf{Res}}_{\mathbf{i-1}}$ the residual network of $\underline{F}_{\mathbf{i-1}}$ for $\underline{\mathbf{f}}_{\mathbf{max,i-1}}$.

We define a *flow shift network* $\mathbf{Shift_i}$ as follows.

Flow shift network: $\mathbf{Shift_i}$ *is a flow network with the same intermediate events of* $\underline{\mathbf{Res}}_{\mathbf{i-1}}$. $\mathbf{Shift_i}$ *has a flow edges* $\mathbf{e_1 \rightarrow e_2}$ *equal to a corresponding one in* $\underline{\mathbf{Res}}_{\mathbf{i-1}}$ *if* $\mathbf{e_1 \neq \{\sigma, \tau\}}$ *and* $\mathbf{e_2 \neq \sigma}$. *Finally, for each edge* $\tau \rightarrow e$ *in* $\underline{\mathbf{Res}}_{\mathbf{i-1}}$ *such that* $e \rightarrow \tau$ *is an auxiliary flow edge in* $\underline{F}_{\mathbf{i-1}}$, $\mathbf{Shift_i}$ *has a corresponding edge* $\sigma \rightarrow e$ *of the same capacity.*

Let us now call call $\mathbf{Res(Shift_i)}$ the residual network of $\underline{F}_{\mathbf{i-1}}$ for $\mathbf{f' = \underline{f}_{max,i} + f_{max,shift,i}}$. We define a *flow reduction network* $\mathbf{Reduce_i}$ as follows.

Flow reduction network: $\mathbf{Reduce_i}$ *is a flow network with the same nodes as* $\mathbf{Res(Shift_i)}$ *and edges* $\mathbf{e_1 \rightarrow e_2}$ *identical to* $\mathbf{Res(Shift_i)}$ *if one of the following three conditions is satisfied:*

1) $\mathbf{e_2 \neq \tau}$;
2) $\mathbf{e_1 \neq \sigma}$;
3) $\mathbf{e_1 = \sigma}$ *and the edge* $\mathbf{\sigma \rightarrow e_2}$ *in* $\mathbf{Shift_i}$ *originates from an auxiliary flow edge for* $\underline{\mathbf{F}}_{\mathbf{i-1}}$.

Using $\mathbf{Shift_i}$ and $\mathbf{Reduce_i}$, we define the **Flow_Contraction** operator needed by the incremental envelope algorithm.

Flow_Contraction$\mathbf{(F(R_{i-1}), f_{max,i-1}, \delta C_i, Aprec)}$**:**

1) *Compute a maximum flow* $\mathbf{f_{max,shift,i}}$ *for* $\mathbf{Shift_i}$;
2) *Compute a maximum flow* $\mathbf{f_{max,red,i}}$ *for* $\mathbf{Reduce_i}$;
3) *Return* $\mathbf{f_{contr,i} = \underline{f}_{max,i} + f_{max,shift,i} + f_{max,red,i}}$

We now prove that the operator keeps the flow maximum.

Lemma 4: *The flow* $\mathbf{f' = \underline{f}_{max,i} + f_{max,shift,i}}$ *is maximum for* $\underline{\mathbf{F}}_{\mathbf{i-1}}$.

Proof: $\mathbf{f'}$ is a flow of $\underline{\mathbf{F}}_{\mathbf{i-1}}$. It is also maximum since by construction of $\mathbf{Shift_i}$ it is $\mathbf{f_{max.shift,i}(\sigma, e) = 0}$. Therefore $\mathbf{f'(\sigma, e) = \underline{f}_{max,i-1}(\sigma, e)}$ and therefore $\mathbf{f'}$ is also maximum for $\underline{\mathbf{F}}_{\mathbf{i-1}}$.

Lemma 5: $\mathbf{f_{contr,i}}$ *is a flow for* $F(\mathbf{R_{i-1} - \delta C_i})$.

Proof: $\mathbf{f_{contr,i}}$ is a flow for $\underline{\mathbf{F}}_{\mathbf{i-1}}$. For it to be a flow for $F(\mathbf{R_{i-1} - \delta C_i})$ it must be $\mathbf{f_{contr,i}(e, \tau) = 0}$ if $\mathbf{e \rightarrow \tau}$ is an auxiliary edge. If it were $\mathbf{f_{contr,i}(e, \tau) > 0}$ for an auxiliary edge, by using the flow conservation constraint we could show that there must be a path from $\boldsymbol{\sigma}$ to $\boldsymbol{\tau}$, passing through $\mathbf{e \rightarrow \tau}$, with all edges having positive flow. Therefore, there must be a flow-reducing path from $\boldsymbol{\tau}$ to $\boldsymbol{\sigma}$ in the corresponding residual network. Such path is an augmenting path in the residual network of $\mathbf{Reduce_i}$ for flow $\mathbf{f_{max,red,i}}$, which contradicts the maximality of $\mathbf{f_{max,red,i}}$.

Theorem 6: $\mathbf{f_{contr,i}}$ *is a maximum flow for* $F(\mathbf{R_{i-1} - \delta C_i})$.

Proof: This is clearly true if $\mathbf{f_{max,red,i}}$ is a null flow since **f'** is maximum. If $\mathbf{f_{max,red,i}}$ is not null, assume that $\mathbf{f_{contr,i}}$ is not maximum. This yields an augmenting path from σ to τ in $\boldsymbol{F}(\mathbf{R_{i-1} - \delta C_i})$ for $\mathbf{f_{contr,i}}$. Since $\underline{\mathbf{f}}_{\mathbf{max,i}}$ is optimal, such path could only have appeared after the computation of $\mathbf{f_{max,shift,i}}$. Since **f'** is maximum for $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$, there must be at least one edge $\mathbf{e_1 \rightarrow e_2}$ on the augmenting path that does not belong to the residual network of $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$ for **f'** while the suffix path from $\mathbf{e_2}$ to τ has positive residual capacity in $\mathbf{Shift_i}$ for $\mathbf{f_{max,shift,i}}$. A positive residual for $\mathbf{e_1 \rightarrow e_2}$ implies that flow reduction pushed flow in the opposite direction, i.e., $\mathbf{f_{max,red,i}(e_2, e_1) > 0}$. Consider the last such edge in the augmenting path. By backtracing its flow we find a positive flow path for $\mathbf{f_{max,red,i}}$ from σ to $\mathbf{e_2}$. This can only happen if the capacity of the path in $\mathbf{Reduce_i}$ is positive, which is equivalent to a prefix path with positive residual capacity in $\mathbf{Shift_i}$ for $\mathbf{f_{max,shift,i}}$. Tying the prefix and postfix at $\mathbf{e_2}$ yields an augmenting path in $\mathbf{Shift_i}$ for $\mathbf{f_{max,shift,i}}$, impossible since $\mathbf{f_{max,shift,i}}$ is maximum.□

Flow Expansion

The completion of stage **i** of the algorithm requires now to incorporate the event set $\delta\mathbf{R_i}$ to yield $\mathbf{R_i}$ and allow the computation of $\mathbf{P_{max,i} = P_{max}(R_i)}$. Again, we define an incremental operation on an incremental residual flow network, the *flow expansion network.* The network is built on the residual network of $\boldsymbol{F}(\mathbf{R_{i-1} - \delta C_i})$ for flow $\mathbf{f_{contr,i}}$. We call this residual network $\mathbf{Res(Contr_i)}$.

Flow expansion network: $\boldsymbol{Expand_i}$ *is a flow network with the intermediate events* $\boldsymbol{R_i}$. $\boldsymbol{Expand_i}$ *all flow edges* $\boldsymbol{e_1 \rightarrow e_2}$ *in* $\boldsymbol{Res(Contr_i)}$, *all flow edges in* $\boldsymbol{F(\delta R_i)}$ *and an infinite capacity edge* $\boldsymbol{e_1 \rightarrow e_2}$ *for each anti-precedence edge between* $\boldsymbol{e_1 \in \delta R_i}$ *and* $\boldsymbol{e_2 \in R_{i-1} - \delta C_i}$.

Note that by construction $\mathbf{Expand_i}$ is the residual network in $\boldsymbol{F}(\mathbf{R_i})$ for $\mathbf{f_{contr,i}}$. We now define the final operator needed by the incremental envelope algorithm, **Flow_Expansion.**

Flow_Expansion($\boldsymbol{F(R_{i-1} - \delta C_i)}$, $\boldsymbol{f_{contr,i}}$, $\boldsymbol{\delta R_i}$, ***Aprec):***

1) Compute a maximum flow $\boldsymbol{f_{max,exp,i}}$ *for* $\boldsymbol{Expand_i}$;

2) Return $\boldsymbol{f_{max,i} = f_{contr,i} + f_{max,exp,i}}$

Theorem 7: $\boldsymbol{f_{max,i}}$ *computed by* ***Flow_Expansion*** *is maximum for* $\boldsymbol{F(R_i)}$.

Proof: $\mathbf{f_{max,i}}$ is clearly a flow for $\boldsymbol{F}(\mathbf{R_i})$. Moreover, $\mathbf{f_{max,exp,i}}$ is maximum for $\mathbf{Expand_i}$ and therefore there is no augmenting path in the corresponding residual network. The maximality of $\mathbf{f_{max,i}}$ follows from the identity between the residual network of $\mathbf{Expand_i}$ for $\mathbf{f_{max,exp,i}}$ and the residual network of $\boldsymbol{F}(\mathbf{R_i})$ for $\mathbf{f_{max,i}}$.□

Flow Separation for $\mathbf{P_{max}}$

We can achieve further performance improvements by minimizing the number of nodes and flow edges that need to be considered at each stage. During stage **i** two $\mathbf{P_{max}}$ are computed: $\mathbf{P_{max,contr,i}}$ after $\mathbf{Flow\_Contraction_i}$ and $\mathbf{P_{max,i}}$ after $\mathbf{Flow\_Expansion_i}$. We know that each $\mathbf{P_{max}}$ is a predecessor set (i.e., it contains all of its successors in the anti-precedence graph), it is flow isolated (i.e., for each pair of events $\mathbf{e_1 \in P_{max}}$ and $\mathbf{e_2 \in P^C_{max}}$, $\mathbf{f_{max}(e_1, e_2) = 0}$ and $\mathbf{f_{max}(e_2, e_1) = 0}$) and has all exit edges saturated (i.e., $\mathbf{f_{max}(e, \tau) = c(e, \tau)}$ for all $\mathbf{e \in P_{max}}$) (Muscettola, 2002). This will allow us to prove that $\boldsymbol{F}(\mathbf{P_{max,i-1}})$ can be ignored during the computation of $\mathbf{Flow\_Contraction_i}$ and $\boldsymbol{F}(\mathbf{P_{max,contr,i}})$ can be ignored during computation of $\mathbf{Flow\_Expansion_i}$.

Let us consider each maximum flow operation executed at stage **i**. The first is flow shifting. Note that by construction, the P_{max} of $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$, $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$, contains the events in $\mathbf{P_{max,i-1} - \delta C_i}$. $\mathbf{P_{max,i-1}}$ is a predecessor set since $\mathbf{\delta C_i}$. contains events at the bottom of the anti-precedence graph for $\boldsymbol{F}(\mathbf{R_{i-1}})$. However, due to the additional links $\mathbf{e \rightarrow \tau}$ the value of the positive residual of $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ is equal to $\Delta(\mathbf{P_{max,i-1}})$. $\mathbf{P_{nax,\ i-1}}$ is still flow insulated and has all exit edges saturated. Assume that flow shifting pumped flow to reach an event $\mathbf{e' \in \underline{P}_{max,i-1}}$. In order for at least part of such flow to reach τ there must be a postfix augmenting path that reaches τ from **e'**. But this is impossible since $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ is a predecessor set, all postfix paths must remain inside $\mathbf{P_{max,i-1}}$, and all exit edges from $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ to τ are saturated. Therefore, any maximum flow algorithm pushing flows that searches for augmenting paths can avoid doing so in $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ and any excess flow pumped into events of $\underline{\mathbf{P}}_{\mathbf{max,i}}$ that can achieve τ will have to be pushed back from $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ to $\underline{\mathbf{P}}^{\mathbf{C}}_{\mathbf{max,,i-1}}$. Therefore we can ignore $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ during flow shifting.

After flow shifting the maximum predecessor set is still $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ since flow shifting simply produces a different maximum flow for $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$ and $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ is independent from the specific flow instance (Muscettola, 2002).

Considering now flow reduction, $\mathbf{f_{max,red,i}}$ this can be computed by simply backtracing flow in $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$. Because of the flow insulation of $\underline{\mathbf{P}}_{\mathbf{max,,i-1}}$, this backtracing is either performed exclusively over events in $\underline{\mathbf{P}}^{\mathbf{C}}_{\mathbf{max,\ i-1}} = \mathbf{P^C_{max,i-1} - \delta C_i}$ or is confined within the events in $\underline{\mathbf{P}}_{\mathbf{max,i.1}} = \mathbf{P_{max,i-1} - \delta C_i}$. Note that since after flow reduction all auxiliary edges must have zero flow, the producers' residual of $\underline{\mathbf{P}}_{\mathbf{max,i-1}}$ after flow contraction must be equal to $\mathbf{\Delta(P_{max,i-1} - \delta C_i)}$.

Finally, we can use a similar argument to the one used for flow shifting to show that $\mathbf{Flow\_Expansion_i}$ can be performed entirely over $\mathbf{F(P^C_{max,contr,i})}$, therefore ignoring $\mathbf{P_{max,,contr,i}}$.

Incremental Computation of $\mathbf{L_{max}}$

We are now ready to derive a recursive equations for the incremental calculation of $\mathbf{L_{max}(t)}$ by transforming Equation 1 through the application of flow reduction and expansion.

From the discussion on flow separation, we know that, after $\mathbf{Flow\_Contraction_{i-1}}$, $\mathbf{P_{max,contr,I} = (P_{max}(R_{i-1}) - \delta C_i) \cup P_{max}(P^c_{max}(R_{i-1}) - \delta C_i)}$. After $\mathbf{Flow\_Expansion_i}$,because of flow separation, we have $\mathbf{P_{max,i} = P_{max,contr,i} \cup P_{max}(P^C_{contr,i} \cup \delta R_i)}$.

Theorem 8: $L_{max}(t)$ *satisfies this recursive equation:*

if $t = t_1$

$L_{max}(t) = \Delta(C_1) + \Delta(P_{max}(R_1))$

if $t = t_i$ *and* $i > 1$

$L_{max}(t) = L_{max}(t_{i-1}) +$ **i**

$\Delta(\delta C_i \cap P^C_{max}(R_{i-1})) +$ **ii**

$\Delta(P_{max}(P^c_{max}(R_{i-1}) - \delta C_i)) +$ **iii**

$\Delta(P_{max}(\delta R_i \cup P^c_{max}(P^c_{max}(R_{i-1} - \delta C_i)))$; **iv**

if $t \neq t_i$, *then*

$L_{max}(t) = L_{max}(t-1)$.

Proof: $\mathbf{L_{max}(t)}$ only changes when $\mathbf{R_t}$ changes, i.e., at a time $\mathbf{t_i}$. Let us consider in turn the application of **Flow_Contraction**$_i$ and **Flow_Expansion**$_i$.

a) **Flow_Contraction**$_i$**:** the level after flow contraction, $\mathbf{L_{max,contr}(t_i)}$ is the weight of the closed events after contraction and of $\mathbf{P_{max,contr,i}}$. Since new events at time $\mathbf{t_i}$ are only closed during flow contraction and $\mathbf{C_i}$ and $\mathbf{P_{max,contr,i}}$ are disjoint, $\mathbf{L_{max,contr}(t_i) = \Delta(C_{i-1} \cup \delta C_i \cup P_{max,contr,i}) = \Delta(C_{i-1} \cup \delta C_i \cup (P_{max}(R_{i-1}) - \delta C_i) \cup P_{max}(P^c_{max}(R_{i-1}) - \delta C_i))}$. Since for any two sets **A** and **B** it is $\mathbf{A \cup (B - A) = B \cup (A - B)}$, with **B** and **(A–B)** being disjoint sets, we have $\mathbf{\delta C_i \cup (P_{max}(R_{i-1}) - \delta C_i) = P_{max}(R_{i-1}) \cup (\delta C_i - P_{max}(R_{i-1}))}$. Since $\mathbf{\delta C_i \subseteq R_{i-1} = P_{max}(R_{i-1}) \cup P^C_{max}(R_{i-1})}$, it is easy to see that $\mathbf{\delta C_i - P_{max}(R_{i-1}) = \delta C_i \cap P^C_{max}(R_{i-1})}$. This yields $\mathbf{L_{max,contr}(t_i) = L_{max}(t_{i-1}) + \Delta(\delta C_i \cap P^C_{max}(R_{i-1})) + \Delta(P_{max}(P^c_{max}(R_{i-1}) - \delta(C_i)))}$, i.e., lines **i, ii** and **iii** in the theorem's statement.

b) **Flow_Expansion**$_i$**:** the only new increment comes from set $\mathbf{P_{max}(P^C_{max,contr,i} \cup \delta R_i) = P_{max}(P^C_{max}(R_{i-1} - \delta C_i) \cup \delta R_i)}$ which yields line **iv** in the theorem's statement.

Algorithm	Time Complexity	Complexity Key
Labeling	O(nmU)	Total pushable flow
Capacity scaling	O(nm logU)	Total pushable flow
Successive shortest paths	$O(n^2m)$	Shortest distance to τ
Generic preflow-push	$O(n^2m)$	Distance label
FIFO reflow-push	$O(n^3)$	Distance label

Table 1: Complexity of maximum flow algorithms

Figure 5 shows the pseudocode of the algorithm. The functions **Flow_Contraction** and **Flow_Expansion** receive as arguments the current flow network $\mathbf{F_{cur}}$, which includes the current maximum flow, the incremental set of events that need to be added/deleted $\mathbf{E_{cur}.\{earliest,latest\}}$, and the anti-precedence graph **Aprec(N)** for the set of all events in the plan **N. Aprec** carries the topological information needed to expand the flow network.

Given the current flow network and its maximum flow both stored in F_{cur}, **Extract_P_max** returns both its maximum increment predecessor set $\mathbf{P_{max}}$ and the restricted network and flow resulting from the elimination of the $\mathbf{P_{max}}$. Comparing with the formula for $\mathbf{L_{max}(t_i)}$ described by

```
Incremental_Resource_Envelope (N, Apred(N))
{ 1: E := { Group events in the input set N into entries E_t with three
            members: a time t and two lists earliest and latest. Event
            e∈N is included in E_t.earliest if et(e) = t and in E_t.latest if
            lt(e) = t. Sort the E_t in increasing order of t. }
  2: L_max := {<-∞, 0>}   /* Maximum resource envelope. */
  3: t_cur := 0;          /* Current time */
  4: L_old := 0;          /* Envelope level at previous iteration. */
  5: L_new := 0;          /* Envelope level at current iteration. */
  6: P_max := ∅;          /* Maximum increment predecessors.*/
  7: F_cur := ∅;          /*Resource increment flow graph with associated
                            maximum flow */
  8: E_cur := ∅;          /* Entry from E at t_cur. */
  9: while (E is not empty)
 10:  {E_cur := pop(E);
 11:   t_cur := E_cur.t;
 12:   L_new := L_old + weight (intersection (Events(F_cur), E_cur.latest));
 13:   F_cur := Flow_Contraction (F_cur, E_cur.latest, Aprec(N));
 14:   <P_max, F_cur> := Extract_P_Max (F_cur);
 15:   L_new := L_new + weight (P_max);
 16:   F_cur := Flow_Expansion (F_cur, E_cur.earliest, Aprec(N));
 17:   <P_max, F_cur> := Extract_P_max (F);
 18:   L_new := L_new + weight (P_max);
 19:   L_max := append (L_max, <t_cur, L_new>);
 20:   L_old := L_new;
       }
  return L_max;
}
```

Figure 5: Incremental envelope algorithm

Theorem 8, line 12 in the algorithm computes **i+ii**, line 15 adds **iii** and line 18 adds **iv**. Note that the algorithm is actually more of a method that can be implemented in different ways depending of which maximum flow algorithm is used in **Flow_Contraction** and **Flow_Expansion**. As we shall see the worst-case time complexity of the method is the same as that the maximum flow algorithm used. We will also see that further optimizations are possible when using specific algorithms such as push-preflow methods.

Complexity Analysis

The complexity analysis of the incremental envelope algorithm applies to a large number of maximum flow algorithms used for **Flow_Contraction** and **Flow_Expansion**. Each algorithm has a *complexity key*, i.e., a measurable entity whose static properties or dynamic behavior during its computations determines its time complexity. Table 1 (adapted from (Ahuja, Magnanti and Orlin, 1992)) reports the time complexity and complexity key of several maximum flow algorithms.

The *Labeling* and *Capacity Scaling* algorithms are based on the original Ford-Fulkerson method. The complexity depends on the strict monotonicity of the flow pushed at each iteration of the algorithm and on the fact that the *total pushable flow* is bound by **nU** where **U** is the maximum

capacity of an edge $\sigma \rightarrow \mathbf{e}$ or $\mathbf{e} \rightarrow \tau$. The *successive shortest paths* class of algorithms is based on the original Edmonds-Karp algorithm. The complexity depends on the fact that flow is pushed through augmenting paths of monotonically increasing length**.** The complexity key for this class of algorithms is the *shortest distance to* τ for each event **e**. For these algorithms it is crucial to demonstrate that the distance function **d(e)** increases by at least one unit after each iteration.

Finally, preflow-push algorithms such as *generic preflow-push* and *FIFO preflow-push* (Goldberg and Tarjan, 1988) maintain a *distance labeling* $\underline{\mathbf{d}}\mathbf{(e).}$ These algorithms use purely local operations that push excess flow available at node $\mathbf{e_1}$ through edges $\mathbf{e_1 \rightarrow e_2}$ such that $\underline{\mathbf{d}}(\mathbf{e_1}) = \underline{\mathbf{d}}(\mathbf{e_2}) + \mathbf{1}$. When excess flow exists at some node and no such edge exist, the node's distance labeling is increased by the minimum amount that re-establishes a one unit difference in distance label over an edge. This allows more flow to be pushed. The complexity of the algorithms depends on creating a valid labeling at each iteration and on the fact that for each node the distance labeling is monotonically increasing up to **2n-1**.

We now analyze the cumulative cost of computing all flows over **2n** stages respectively for $\mathbf{f_{max,shift,i}}$, $\mathbf{f_{max,red,i}}$ and $\mathbf{f_{max,exp,i}}$**.** First note that at each stage $\mathbf{f_{max,red,i}}$ can be computed by flow backtracing through a backwards depth first search on the resource increment flow network. Since this can cost up to **O(m),** the total cost of computing flow reduction is **O(nm)** and is therefore smaller than the cost of applying a regular maximum flow algorithm. Therefore we focus on the cost for the cumulative $\mathbf{f_{max,shift,i}}$ and $\mathbf{f_{max,exp,i}}$, respectively $\mathbf{F_{shift}} = \mathbf{\Sigma_i}\ \mathbf{f_{max,shift,i}}$ and $\mathbf{F_{exp}} = \mathbf{\Sigma_i}\ \mathbf{f_{max,exp,i}}$.

Lemma 9: *Each of* $\boldsymbol{F_{shift}}$ *and* $\boldsymbol{F_{exp}}$ *is no greater than* $\boldsymbol{n\ U}$.

Proof: Consider $\mathbf{F_{shift}}$ (the argument is similar for $\mathbf{F_{exp}}$). The upper bound of the total capacity of the edges $\sigma \rightarrow \mathbf{e}$ entering $\mathbf{Shift_i}$ is the total capacity of edges $\mathbf{e^- \rightarrow \tau}$ with $\mathbf{e^- \in \delta(C_i)}$. After iteration **i** all nodes in $\mathbf{\delta C_i}$ are eliminated from further consideration, hence flow can go through each $\sigma \rightarrow \mathbf{e}$ only during iteration **i**. Therefore, the total flow is upper bounded by $\Sigma_i\ |\mathbf{\delta C_i}|\ \mathbf{U} = \mathbf{n\ U}$.□

Note that the argument above does not hold for $\mathbf{F_{exp}}$ if instead of using flow shifting the flow is simply reduced and then expanded again (Kumar, 2003). In this case the same flow could be pushed up to **n** times with $\mathbf{F_{exp}}$ being $\mathbf{O(n^2U)}$. This would not improve on the staged envelope algorithm.

Consider now the distance **d(e)** for node **e** and how it changes when computing $\mathbf{f_{max,shift,i}}$.and $\mathbf{f_{max,exp,i}}$. Let us call $\mathbf{d^0_{shift,i}(e)}$ and $\mathbf{d^f_{shift,i}(e)}$ the distances at the beginning and at the end of flow shifting for iteration **i.** We define similarly $\mathbf{d^0_{exp,i}(e)}$ and $\mathbf{d^f_{exp,i}(e)}$.

Lemma 10: $\boldsymbol{d^f_{exp,i-1}(e) \le d^0_{shift,i}(e)}$ *and* $\boldsymbol{d^f_{shift,i}(e) \le d^0_{exp,i}(e)}$.

Proof: Between the end of flow expansion at iteration **i-1** and the start of flow shifting at iteration **i,** the auxiliary flow network changes through the elimination of nodes and edges in $\boldsymbol{F}\ \mathbf{(\delta C_i)}$. Therefore, the new distances in the remaining residual capacity network can only increase. Since $\mathbf{Shift_i}$ only adds edges $\sigma \rightarrow \mathbf{e,}$ the distances in $\mathbf{Shift_i}$ must analogously increase and therefore $\mathbf{d^f_{exp,i-1}(e)} \le \mathbf{d^0_{shift,i}(e).}$ For $\mathbf{Expand_1}$ node distances can further increase because flow reduction can only eliminate residual network edges present in $\mathbf{Shift_i}$ for $\mathbf{f_{max,shift,i}}$**.** Also, from Lemma 3 the addition of F $(\mathbf{\delta R_i})$cannot reduce distances since it cannot add any edge from an event in $\mathbf{Shift_i}$ to one in $\mathbf{\delta R_{t(i)}}$. Therefore, $\mathbf{d^f_{shift,i}(e) \le d^0_{exp,i}(e)}$.□

Note that the argument in Lemma 10 does not hold if events are added in arbitrary order. In this case the addition of edges can reduce the distance function of some node **e** between a shifting and an expansion phase. In the worst case, this may reduce the distance to one for each application of maximum flow and therefore does not improve on the staged algorithm.

Finally, consider reusing distance labeling across preflow-pushes for shifting and expansion. $\underline{\mathbf{d}}^{\mathbf{0}}_{\mathbf{shift,i}}$, $\underline{\mathbf{d}}^{\mathbf{f}}_{\mathbf{shift,i}}$, $\underline{\mathbf{d}}^{\mathbf{0}}_{\mathbf{exp,i}}$ and $\underline{\mathbf{d}}^{\mathbf{f}}_{\mathbf{exp,i}}$ are the distance labelings at the beginning and end of shifting and expansion. Assume also that the distance label of a node that has not yet entered $\mathbf{Expand_i}$ or $\mathbf{Shift_i}$ is zero.

Lemma 11: $\boldsymbol{\underline{d}^0_{shift,I}}$ *can be made equal to* $\boldsymbol{\underline{d}^f_{exp,i-1}}$ *for all nodes in* $\boldsymbol{Shift_i}$**.** *Also,* $\boldsymbol{\underline{d}^0_{exp,i}}$ *can be made equal to* $\boldsymbol{\underline{d}^f_{shift,I}}$ *for all nodes in* $\boldsymbol{Expand_i}$.

Proof: The distance label of a node remains valid when edges are deleted or new edges are only added to enter it from new nodes. Also, a distance function at node **e** must be an upper bound of its labeling. From Lemma 10 we know that the distance function can only increase from $\mathbf{Expand_{i-1}}$ to $\mathbf{Shift_i}$ and from $\mathbf{Shift_i}$ to $\mathbf{Expand_i}$. Therefore $\underline{\mathbf{d}}^{\mathbf{f}}_{\mathbf{exp,i-1}}$ and $\underline{\mathbf{d}}^{\mathbf{f}}_{\mathbf{shift,i}}$ are valid choices respectively for $\underline{\mathbf{d}}^{\mathbf{0}}_{\mathbf{shift,i}}$ and $\underline{\mathbf{d}}^{\mathbf{0}}_{\mathbf{exp,i.}}$.□

We can now prove the main complexity result.

Theorem 12: *For a large class of maximum flow algorithms,* ***Incremental_Resource_Envelope*** *has time complexity* $\boldsymbol{O(Maxflow(n, m, U))}$.

Proof: Assume we applied one of the maximum flow algorithms in Table 1 to find the full maximum flow on the entire resource increment flow network (e.g., to compute the maximum envelope level over the entire time horizon (Muscettola, 2002)) with cost **O(Maxflow(n, m, U))**. We use Lemmas 9, 10 and 11 to prove that the cumulative cost of using the same algorithms for flow shifting and flow expansion is **O(Maxflow(n, m, U))** for the same algorithm.

1. *Labeling and Capacity scaling*: Lemma 9 shows that the worst case bound for the total flow moved during shifting and expansion is at worst twice that for full flow calculation. Also, at each iteration during shifting and expansion, finding an augmenting path costs at most **m** as for full flow calculation. Hence shifting and expansion cost at most **O(Maxflow(n, m, U)).**

2. *Successive shortest paths*: the cost bound for each full flow augmentation is an upper bound for that in shifting and expansion. The algorithm's complexity also depends on the monotonic increase of the distance function up to **n** after each elementary operation. Note that until the deletion of a δC_i or a P_{max}, a node's distance is bound by **n** as for the full flow. Monotonic increase is guaranteed by the algorithm within each shifting and expansion phase and by Lemma 10 across phases. Hence, the cost is **O(Maxflow(n, m, U))**.
3. *Preflow-push methods:* the complexity is found through amortized analysis (Goldberg and Tarjan, 1988), relying on an appropriate potential function Φ and on the determination of its possible variations after the applying a local operation (e.g., a saturating or a non-saturating preflow push). One key observation is the monotonic increase of each node's distance label for each local operation. Both for the incremental and for the full flow this increase is bound by **2n–1** and Lemma 11 guarantees monotonic distance label increase across phases. Note that, unlike for the full flow, for shifting and expansion Φ increases also at the beginning of each shifting phase, when nodes are activated by the creation of initial flow excesses. However, a detailed amortized analysis (omitted for space limitations) shows that this increase does not affect the order of complexity of the shifting and expansion phases that remains **O(Maxflow(n, m, U))**.

The worst case complexity of the other phases of ***Incremental_Resource_Envelope*** besides shifting and expansion are dominated by **O(Maxflow(n, m, U)**. Flow reduction is cumulatively **O(nm).** The total cost of **Extract_P_max** and of incrementally constructing and deleting the flow network**,** is **2 O(m).** Finally the sorting of events during initialization is **O(n log n).**□

Optimized Preflow-Push Implementation

If the incremental algorithm is implemented using a preflow-push method, the previous complexity analysis indicates that, in order to reduce complexity, we need to reuse the distance labeling function from the end of a maximum flow computation to the start of the next.

A further optimization is possible. Consider the maximum flow calculation on $\mathbf{Shift_i}$. During initialization, an excess flow is loaded on each event **e** for each edge $\sigma \rightarrow \mathbf{e}$ in $\mathbf{Shift_i}$. We know that only a fraction of this excess flow may reach τ. The remainder will be pushed back out of $\mathbf{Shift_1}$ during flow shifting and then pushed again through the flow network during flow reduction. In other words, this flow travels *twice* through the network before being eliminated. We can remove this duplication as flows. Assume that instead of deleting the $\sigma \rightarrow \mathbf{e}$ edges of $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$ when constructing $\mathbf{Shift_i}$ we delete instead the $\sigma \rightarrow \mathbf{e}$ edges of $\mathbf{Shift_i}$ after having performed the appropriate excess loading needed to perform flow shifting. In this case the flow that cannot be shifted will be pushed back to the source in $\underline{\boldsymbol{F}}_{\mathbf{i-1}}$, i.e., $\boldsymbol{F}(\mathbf{R_i})$, instead making the additional **O(nm)** cost of flow reduction unnecessary. Another possible optimization consists of combining preflow-push through $\mathbf{Shift_i}$ and $\mathbf{Expand_i}$ by connecting $\delta \mathbf{R_i}$ *before* running the shift/reduce preflow-push. In this case the flow excess initializations of contraction and expansion are combined and a single preflow-push is run during phase **i**.

These optimizations do not affect asymptotic complexity but may have a significant effect in practice. Empirical studies will be needed to assess their actual usefulness.

Conclusions

We presented a new algorithm that efficiently computes resource envelopes for flexible plans. The methods has complexity **O(Maxflow(n, m, U))** where **n** and **m** measure the size of the activity plan and **U** measures the maximum resource consumption or production of an activity. We are currently experimenting with various implementations of the methods. While we expect that for large problem sizes the **O(n)** cost reduction will be evident, practical improvements on smaller problems require careful design of efficient and minimal data structures. We are also addressing the second part of the cost/benefit equation for envelopes by exploiting additional structural information resulting from the method's incrementality and designing scheduling algorithms that use a minimal search space.

References

R.K. Ahuja, T.L. Magnati, J.N. Orlin, 1993. *Network Flows,* Prentice Hall, NJ.

R.K. Ahuja, M. Kodialam, A.K. Mishra, J.B. Orlin, 1997. Computational Investigations of Maximum Flow Algorithms, *European Journal of OR,* Vol 97(3).

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

A.V. Goldberg, R.E. Tarjan, 1988. A New Approach to the Maximum-Flow Problem. *JACM*, Vol. 35(4).

T.K.S. Kumar, 2003. Incremental Computation of Resource-Envelopes in Producer-Consumer Models, *Procs. of CP2003,* Kinsale, Ireland.

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

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

N. Muscettola, 2002. Computing the Envelope for Stepwise-Constant Resource Allocations, *Procs. of CP2002,* Ithaca, NY.

N. Pollicella, S.F. Smith, A. Cesta, A. Oddi, 2003. Steps toward Computing Flexible Schedules, *Procs. of Online-2003 Workshop at CP 2003,* Kinsale, Ireland, http://www.cs.ucc.ie/~kb11/CP2003Online/onlineProceedings.pdf