

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 executions of a temporally flexible plan. We present a new class of algorithms that computes an envelope in $O(\text{Maxflow}(n, m, U))$ where n , m and U measure the size of the flexible plan. This is an $O(n)$ improvement on the best complexity bound for an envelope algorithm known so far and makes envelopes more amenable to practical use in scheduling. 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 used 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, 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 comparative studies of scheduling algorithms using envelopes appear not to show a computational advantage with respect to using more traditional heuristic methods based on fixed-time resource profiles (Pollicella et al., 2003). Since computing envelopes is more computationally expensive than building a fixed-time profile, it is critical to ensure that the balance between computation cost and increased structural information extracted from the envelope is advantageous. 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. Currently, the resource envelope algorithm known to have the best asymptotic complexity (Muscettola, 2002) computes all piecewise-constant segments of the envelope through as many as $2n$ stages, where n is the number of events (start or end of activities) in the flexible plan. Each stage computes a maximum flow and therefore the overall complexity of the method is $O(n \text{Maxflow}(n, m, U))$ 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 $\text{Maxflow}(n, m, U)$ is the asymptotic cost of the maximum flow algorithm.

This staged method, however, can be significantly improved since at each stage a full maximum flow for the entire flexible plan is recomputed from scratch. Cost reduction could be obtained through an incremental flow method. Starting from the maximum flow at one stage, the maximum flow for the next stage is obtained 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 $O(\text{Maxflow}(n, m, U))$ 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 $O(n^{1.5})$ (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 (Muscatto, 2002). Next we present the new incremental algorithm and identify all sources of performance improvements. We then prove the complexity result, discuss implementation improvements when using preflow-push algorithms and conclude by discussing future work.

Staged Computation of Envelopes

In this section we outline the envelope problem and the staged algorithm that solves it. For a complete discussion, see (Muscatto, 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., e_{1s} and e_{1e} for activity A_1), a non-negative flexible activity duration link (e.g., $[2, 5]$ for activity A_1), and flexible separation links between events (e.g., $[0, 4]$ from e_{3e} to e_{4s}). Two additional events T_s and 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 also has an *allocation value* $r(e)$ (e.g., $r(e_{3s}) = -2$), 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 e^- is a *consumer*, if it is positive e^+ is a *producer*. We assume that the temporal constraints are consistent which means that for any pair of events the shortest path $|e_1, e_2|$ from e_1 to e_2 is well defined. Each event e can occur within its time bound, between the earliest time $et(e) = -|e, T_s|$ and the latest time $lt(e) = |T_e, e|$. The triangular inequality $|e_1, e_3| \leq |e_1, e_2| + |e_2, e_3|$ holds for any three events e_1, e_2 and e_3 .

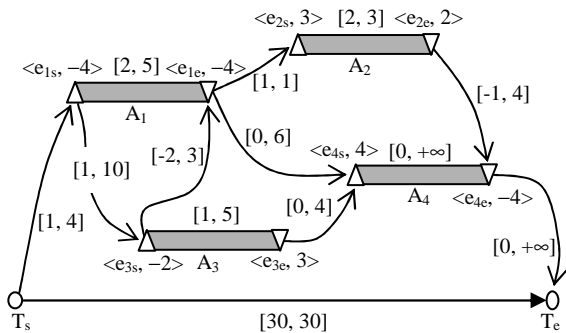


Figure 1: An activity network with resource allocations

Informally, a flexible plan is resource consistent if the duration and separation links induce appropriate necessary precedence relations between consumers and producers. These relations should guarantee that, when a consumer occurs, the total resource level due to consumers and producers that cannot occur after it must be at least as high

as the new consumption. A similar condition applies to the correct occurrence of a producer. The full information on the necessary precedence relations is captured the *anti-precedence graph* A_{prec} , a graph that contains a path between any two events e_1 and e_2 if and only if $|e_1, e_2| \leq 0$. 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 use anti-precedence graphs rather than the most customary precedence graphs (Laborie, 2001) to simplify the construction of the auxiliary maximum flow problem that, as we will see, is fundamental for the computation of envelopes.

We can now formally define a resource envelope. For any subset of events A , the *resource level increment* of A is $\Delta(A) = 0$ if $A = \emptyset$, and $\Delta(A) = \sum_{e \in A} r(e)$ if $A \neq \emptyset$. 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 $s \in S$ is $L_s(t) = \Delta(E_s(t))$. Here $E_s(t)$ is the set of events e which occur at or before t in s . The *maximum resource envelope* is $L_{max}(t) = \max_{s \in S} L_s(t)$ and the *minimum resource envelope* is $L_{min}(t) = \min_{s \in S} L_s(t)$.

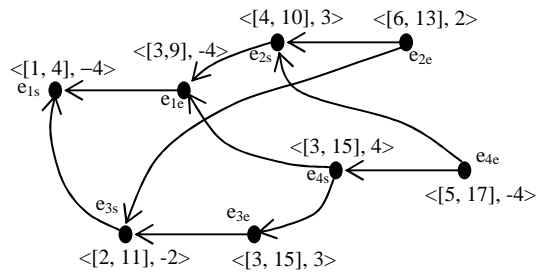


Figure 2: Anti-precedence graph with time bounds and resource allocations

$= \min_{s \in S} L_s(t)$. Since L_{min} can be computed with obvious term substitution on the method that computes L_{max} , we only focus on 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* C_t that must occur before or at t , i.e., such that that $lt(e) \leq t$; 2) the *pending events* R_t that can occur before, at or after t , i.e., such that $et(e) \leq t < lt(e)$; and 3) the *open events* O_t that must occur strictly after t , i.e., such that $et(e) > t$.

Any resource level increment $L_s(t)$ will always include the contribution of all events in C_t and none of those in O_t , but may include only some subset of events in 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* $P \subseteq R_t$, such that if $e \in P$ and e' follows e in A_{prec} , then $e' \in P$. We call $P_{max}(R_t)$ the (possibly empty) predecessor set with maximum non-negative resource level increment.

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

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

For completeness, we note that $F(\delta C_i)$ consists of node e_{1s} and edge $e_{1s} \rightarrow \tau$ while $F(\delta R_i)$ consists of node e_{2s} and edge $\sigma \rightarrow e_{2s}$. All other added and deleted edges are connectives from $F(R_{i-1} - \delta C_i)$ to $F(\delta C_i)$ (edges $e_{1e} \rightarrow e_{1s}$ and $e_{3s} \rightarrow e_{1s}$) and from $F(\delta R_i)$ and $F(R_{i-1} - \delta C_i)$ (edge $e_{2s} \rightarrow e_{1s}$).

The sets δC_i and δR_i satisfy the following fundamental properties.

Lemma 3: δC_i is a predecessor set contained in R_{i-1} . δR_i is the complement of predecessor set R_{i-1} in R_i .

Proof: We only give the proof for δC_i since the one for δR_i is analogous. We need to show that no predecessor of an event in δC_i can belong to its complement in R_{i-1} . In other words, given $e_1 \in \delta C_i$ and $e_2 \in R_{i-1} - \delta C_i$, using the definition of anti-precedence graph and predecessor set, it must be $|e_1 e_2| > 0$. From the definition of δC_i we have $lt(e_1) = t_i$ and $lt(e_2) \geq t_i + 1$. From the triangular inequality applied to the latest times of e_1 and e_2 , $lt(e_2) \leq lt(e_1) + |e_1 e_2|$, we deduce $|e_1 e_2| \geq lt(e_2) - lt(e_1) \geq t_i + 1 - t_i = 1 > 0$. \square

Lemma 3 determines what flow edges are eliminated when δC_i is deleted and what are added when δR_i is added. In particular, we can only delete edges that enter events in δC_i or go from δC_i to τ . Similarly, we can only add edges that exit events in δR_i or go from σ to δR_i . Unlike previous proposals for incremental envelope calculation (Kumar, 2003), our method 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.

Directly related to Lemma 3 is the possibility of computing the maximum flow of $F(R_i)$ by incrementally modifying the flow of $F(R_{i-1})$, reusing both flow values and intermediate data structures across successive invocations of a maximum flow algorithm. We will prove that our flow modification operators guarantee the maximality of each intermediate flow. Maintaining intermediate flow maximality and reusing data structures 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 P_{max} is used in the envelope calculation, $F(P_{max})$ and all of its connecting edges can be safely eliminated from further consideration. This reduces flow network size and further contributes to cost reduction.

The argument to construct the fast envelope algorithm will proceed as follows. On the basis of Lemma 3, we first define flow transformation operators that apply to network additions/deletions occurring between time t_{i-1} and time t_i . The operators define the sequence of maximum-flow problems that need to be solved. Then we identify a flow separation property that, once applied to the sequence of maximum flow problems, further reduces the size of each step's maximum flow problem. Finally, we determine a

¹ For ease of exposition we assume discrete time although the theory applies also to continuous time with appropriate modifications.

recursive equation that computes $L_{max}(t_i)$ as a function of $L_{max}(t_{i-1})$ and the weights of event sets deduced from the application of flow transformation operators.

Flow Modification Networks

The philosophy of each flow transformation 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 f , an operator first defines an auxiliary flow transformation network F_T , then finds one of its maximum flows f_T and finally produces a flow $f_{new} = f + f_T$. Each F_T consists of selected edges in the residual network of F for f . Since the properties of flows are preserved in the sum of a flow of F and a flow of its residual network, f_{new} is also a flow for network F .

Consider now the resource increment flow network $F(R_{i-1})$ at stage $i-1$ and assume that the set of new closed events δC_i is not empty. At stage i all events in δC_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 δC_i will necessarily have to be zeroed, i.e., pushed back into $F(R_{i-1})$. The value of this flow is the sum of the residual capacities of all edges $e_1 \rightarrow e_2$ where $e_1 \in \delta C_i$ and $e_2 \in R_{i-1} - \delta C_i$. When pushed back, this flow can follow two routes. The first reaches τ through some non-saturated exiting edges of $F(R_{i-1} - \delta C_i)$. If after having followed the first route some flow is still flowing on some $e_1 \rightarrow e_2$ but the flow cannot reach τ any more, a second route allows reversing the remaining flow all the way to σ . We call this flow push-back operation a *flow contraction*. The first flow route corresponds to a *flow shift* and the second one to a *flow reduction*. For example, consider the network in Figure 4. Assume that at $t=3$ it is $f_{max}(e_{1s}, \tau) = 4$, $f_{max}(e_{1e}, \tau) = 1$ and $f_{max}(e_{3s}, \tau) = 2$. At $t=4$ the elimination of e_{1s} requires pushing back 4 units of flow. Three of these units can still reach τ by being shifted to $e_{1e} \rightarrow \tau$. Only one unit of flow needs to be pushed back to σ . If we pushed four units of flow back to σ without shifting (as in (Kumar, 2003)), later we would need to push again three units of flow from σ to τ to ensure flow maximality. This repetition of work affects worst-case asymptotic complexity.

Assume now that at stage i there is also a non-empty set δR_i of new pending events. Augmenting $F(R_{i-1} - \delta C_i)$ with the part of the resource increment flow network pertaining to δR_i yields $F(R_i)$. Assume now that $F(R_{i-1} - \delta C_i)$ is traversed by the flow resulting from flow contraction. Even if this flow is maximum for $F(R_{i-1} - \delta C_i)$, in general it will not be maximum for $F(R_i)$ since additional flow could be pushed through edges $\sigma \rightarrow e$ with $e \in \delta R_i$. 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 $F(R_i)$ by moving a minimal amount of flow.

Flow Contraction

Flow expansion network: $Expand_i$ is a flow network with nodes corresponding to those of R_i . $Expand_i$ contains all flow edges $e_1 \rightarrow e_2$ in $Res(Contr_i)$, all flow edges in $F(\delta R_i)$ and an infinite capacity edge $e_1 \rightarrow e_2$ for each anti-precedence edge between $e_1 \in \delta R_i$ and $e_2 \in R_{i-1} - \delta C_i$.

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

Flow_Expansion($F(R_{i-1} - \delta C_i)$, $f_{contr,i}$, δR_i , $Aprec$):
 1) Compute a maximum flow $f_{max,exp,i}$ for $Expand_i$;
 2) Return $f_{max,i} = f_{contr,i} + f_{max,exp,i}$

Theorem 7: $f_{max,i}$ is maximum for $F(R_i)$.

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

Flow Separation for 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 P_{max} are computed: $P_{max,contr,i}$ after **Flow_Contraction_i**, and $P_{max,i}$ after **Flow_Expansion_i**. We know that each 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 $e_1 \in P_{max}$ and $e_2 \in P_{max}^c$, $f_{max}(e_1, e_2) = 0$ and $f_{max}(e_2, e_1) = 0$) and has all exit edges saturated (i.e., $f_{max}(e, \tau) = c(e, \tau)$ for all $e \in P_{max}$) (Muscettola, 2002). This allows us to prove that $F(P_{max,i-1})$ can be ignored during the computation of **Flow_Contraction_i**, and $F(P_{max,contr,i})$ can be ignored during the computation of **Flow_Expansion_i**.

Let us consider each maximum flow operation executed at stage i . The first is flow shifting. Let us consider the properties of the set $P_{max,i-1}$ of F_{i-1} that contains the events in $P_{max}(R_{i-1}) - \delta C_i$. $P_{max,i-1}$ is a predecessor set since all events in δC_i are at the bottom of the anti-precedence graph for $F(R_{i-1})$. Moreover, due to added auxiliary edges in F_{i-1} , the residual of the producers of $P_{max,i-1}$ is the same as that in $P_{max,i-1}$ and therefore equal to $\Delta(P_{max}(R_{i-1}))$. $P_{max,i-1}$ is still flow insulated and has all exit edges saturated. Assume that at some point during the flow shifting operation some additional flow reached an event $e' \in P_{max,i-1}$. 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, being $P_{max,i-1}$ a predecessor set, all postfix paths must remain inside $P_{max,i-1}$ and all exit edges from $P_{max,i-1}$ to τ are saturated. Therefore, any maximum flow algorithm that searches for augmenting paths can avoid doing so in $P_{max,i-1}$. Moreover, in order for any excess flow pumped into events of $P_{max,i}$ to achieve τ , that flow will have to be pushed back from $P_{max,i-1}$ to $P_{max,i-1}^c$. Therefore we can ignore $P_{max,i-1}$ during flow shifting.

Since after flow shifting no flow has been changed for edges that touch $P_{max,i-1}$, $P_{max,i-1}$ maintains the flow insulation and saturation properties it had before flow shifting.

Considering now flow reduction, $f_{max,red,i}$ this can be computed by simply back-tracing flow in F_{i-1} . Because of the flow insulation of $P_{max,i-1}$, this back-tracing is either performed exclusively over edges connecting events in $P_{max,i-1}^c = P_{max}^c(R_{i-1}) - \delta C_i$ or is confined within edges connecting events in $P_{max,i-1}$. Since the entire flow that exits $P_{max,i-1}$ will be back-traced, the entire $P_{max,i-1} = P_{max}(R_{i-1}) - \delta C_i$ must belong to $P_{max,contr,i}$, the maximum predecessor set obtained after **Flow_Contraction_i**. Therefore the contribution of $P_{max,i-1}$ to $P_{max,contr,i}$ can be known in advance without having to modify any flow of $F(P_{max,i-1})$ during **Flow_Contraction_i**. Hence, $P_{max}(R_{i-1})$ can be taken out of consideration for future flow calculation as soon as it is computed.

Finally, we can use a similar argument to the one used for flow shifting to show that **Flow_Expansion_i** can be performed entirely over $F(P_{max,contr,i}^c)$, therefore allowing us to ignore $P_{max,contr,i}$ at any future stage.

Incremental Computation of L_{max}

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

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

if $t = t_i$,
 $L_{max}(t) = \Delta(C_i) + \Delta(P_{max}(R_i))$
 if $t = t_i$ and $i > 1$
 $L_{max}(t) = L_{max}(t_{i-1}) + \Delta(\delta C_i \cap P_{max}(R_{i-1})) + \Delta(P_{max}(P_{max}^c(R_{i-1}) - \delta C_i)) + \Delta(P_{max}(\delta R_i \cup P_{max}^c(P_{max}(R_{i-1}) - \delta C_i)))$; *iv*

if $t \neq t_i$, then

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

Proof: $L_{max}(t)$ only changes when R_i changes, i.e., at a time t_i . Consider in turn the application of **Flow_Contraction_i** and **Flow_Expansion_i**. Because of flow separation after **Flow_Contraction_{i-1}**, we have $P_{max,contr,i} = (P_{max}(R_{i-1}) - \delta C_i) \cup P_{max}(P_{max}^c(R_{i-1}) - \delta C_i)$. Analogously, after **Flow_Expansion_i**, we have $P_{max,i} = P_{max,contr,i} \cup P_{max}^c(P_{max,contr,i} \cup \delta R_i)$.

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

