Incremental Computation of Resource-Envelopes in Producer-Consumer Models

T. K. Satish Kumar

Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

Abstract. Interleaved planning and scheduling employs the idea of extending partial plans by regularly heeding to the scheduling constraints during search. One of the techniques used to analyze scheduling and resource consumption constraints is to compute the so-called *resource-envelopes*. These envelopes can then be used to derive effective heuristics to guide the search for good plans and/or dispatch given plans optimally. The key to the success of this approach however, as in the case of interleaved planning and execution monitoring (where fast reasoning about unintended outcomes is extremely important), is in being able to recompute the envelopes incrementally as and when partial commitments are made. The resource-envelope problem in producer-consumer models is as follows: A directed graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ has $\mathcal{X} = \{X_0, X_1 \ldots X_n\}$ as the set of nodes corresponding to events (X_0 is the "beginning of the world" node and is assumed to be set to 0) and $\mathcal{E}$ as the set of directed edges between them. A directed edge $e = \langle X_i, X_j \rangle$ in $\mathcal{E}$ is annotated with the simple temporal information $[LB(e), UB(e)]$ indicating that a consistent schedule must have X_j scheduled between $LB(e)$ and $UB(e)$ seconds after X_i is scheduled. Some nodes (events) correspond physically to production or consumption of resources and are annotated with a real number $r(X_i)$ indicating their level of production or consumption of a given resource. Given a consistent schedule s for all the events, the total production (consumption) by time t is given by $P_s(t)$ ($C_s(t)$). The goal is to build the envelope functions $g(t) = max_{\{s \text{ is a consistent schedule}\}}(P_s(t) - C_s(t))$ and $h(t) = min_{\{s \text{ is a consistent schedule}\}}(P_s(t) - C_s(t))$. In this paper, we provide efficient incremental algorithms for the computation of $g(t)$ and $h(t)$, along with flexible consistent schedules that actually achieve them for any given time instant t.

1 Introduction

Interleaved planning and scheduling employs the idea of extending partial plans by regularly heeding to the scheduling constraints during search. One of the techniques used to analyze scheduling and resource consumption constraints in the context of the currently maintained partial plan is to compute the so-called resource-envelopes. These can then be used to guide the search for a good plan in a variety of ways (see [2] and [4]).

First, they provide sanity checks for early backtracking when it is possible to examine the envelopes and determine that no consistent schedule for the current set of constraints could possibly satisfy all the resource constraints. Second, they provide a heuristic value for estimating the *constrainedness* of a partial plan. Third, they provide a search termination criterion when it is possible to examine the envelopes and determine that any consistent schedule for the current set of constraints would be successful in satisfying the resource constraints. Fourth, they provide important subroutines for designing approximation algorithms in optimal dispatching/scheduling of plans.

The resource-envelope problem in producer-consumer models is as follows: A directed graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ has $\mathcal{X} = \{X_0, X_1 \ldots X_n\}$ as the set of nodes corresponding to events (X_0 is the "beginning of the world" node and is assumed to be set to 0) and $\mathcal{E}$ as the set of directed edges between them. A directed edge $e = \langle X_i, X_j \rangle$ in $\mathcal{E}$ is annotated with the simple temporal information $[LB(e), UB(e)]$ indicating that a consistent schedule must have X_j scheduled between $LB(e)$ and $UB(e)$ seconds after X_i is scheduled. Some nodes (events) correspond physically to production or consumption of resources and are annotated with a real number $r(X_i)$ indicating their level of production or consumption of a given resource. Given a consistent schedule s for all the events, the total production (consumption) by time t is given by $P_s(t)$ ($C_s(t)$). The goal is to build the envelope functions $g(t) = max_{\{s \text{ is a consistent schedule}\}}(P_s(t) - C_s(t))$ and $h(t) = min_{\{s \text{ is a consistent schedule}\}}(P_s(t) - C_s(t))$.

The producer-consumer model captures several realities associated with reasoning about actions and plans. A partial plan typically consists of a set of actions, a set of open conditions (sub-goals that still need to be achieved), a set of established causal links, a set of temporal constraints between various events that include the beginning and end points of actions, and a set of resource requirements associated with the execution of individual actions (see [6]). An action A can consume (produce) a resource in a variety of ways that include: (1) A holding w_A amount of resource at the beginning of its execution and returning it at the end, (2) A holding w_A amount of the resource at the beginning of its execution and not returning it at the end, (3) A producing w_A amount of the resource at the end of its execution etc. All these can be expressed using the producer-consumer model and although continuous consumption and production of resources during execution of individual actions needs to be handled in a more general framework, the producer-consumer model is fairly expressive and the techniques that are shown in this paper to analyze them are illustrative of more complex resource production and consumption models.

Some attempts for computing the envelopes in producer-consumer models have been made in [2] and [4]. This paper improves upon them in a number of ways. First, the estimation of the envelopes provided in [2] is conservative, while it is tight in [4] and in the algorithms provided in this paper. Tightness in the estimates of $g(t)$ and $h(t)$ is extremely important because a tight bound can save us a potentially exponential amount of search through early backtracking and solution detection when compared to a looser bound. Tight bounds also provide

better heuristic estimates for the *constrainedness* of a problem during search. Second, our algorithms are constructive in the sense that we can determine a flexible schedule s that actually achieves $g(t)$ or $h(t)$ for any given time instant t. This ensures good performance at bottle-neck points (as argued in [3]) and is better than determining an arbitrary fixed schedule because flexible schedules tend to be robust in dealing with exogenous events and uncertainty of execution. Third, and most important, our algorithms are incremental—that is, they effectively reuse computation for determining the envelopes as and when search proceeds by making partial commitments and refining plans. Incremental computation is extremely important because the envelopes must be computed at every point in the search space and even a small saving in the complexity saves us an exponential amount of work. In the context of interleaved planning and execution monitoring for example, execution of a (partial) plan may not always result in the intended outcome and fast re-planning is necessary. Incremental computation of resource envelopes then becomes extremely important for an active management of planning and execution monitoring. We show that the incremental complexity of our algorithms is significantly lesser than re-computation from scratch—hence saving us a total amount of work that is proportional to this difference times the size of the search space. Fourth, we show how our algorithms can be adapted to reuse computation even within a single instance of the problem when envelopes need to be recalculated at discontinuities.

Throughout the paper, we will concentrate on (incremental) algorithms for computing the envelope function $g(t)$ only. The computation of $h(t)$ can be done in a directly analogous fashion by simply reversing the role of consumers and producers. This is because $h(t) = min_{\{s \text{ is a consistent schedule}\}}(P_s(t) - C_s(t)) = -max_{\{s \text{ is a consistent schedule}\}}(C_s(t) - P_s(t))$. We will assume that the temporal constraints specified in $\mathcal{E}$ are consistent [1] and will denote the set of all production events (all events $u \in \mathcal{X}$ such that $r(u) > 0$) by $\mathcal{P}$ and the set of all consumption events (all events $u \in \mathcal{X}$ such that $r(u) < 0$) by $\mathcal{C}$.

2 Computing the Envelopes

In this section, we provide efficient algorithms for computing the profile function $g(t)$ given an instance of the resource-envelope problem. Figure 1 shows the algorithm for computing $g(t)$ at a specified time instant t, along with a flexible consistent schedule s that actually achieves it. Figure 2 shows the algorithm for computing $g(t)$ for all t and Figure 3 shows a small example. A series of Lemmas are presented that prove the correctness of the algorithms.

Lemma 1: A consistent schedule exists for $X_1, X_2 \ldots X_n$ in $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$ if and only if $\mathcal{D}(\mathcal{G})$ does not have any negative cycles (see step 1 of Figure 1).

Proof: If there is a negative cycle $X_{i_1}, X_{i_2} \ldots X_{i_k}$, the following are true of the constraints specified in $\mathcal{E}$: $X_{i_2} - X_{i_1} \leq w_1, X_{i_3} - X_{i_2} \leq w_2 \cdots X_{i_k} - X_{i_{k-1}} \leq w_{k-1}$, $X_{i_1} - X_{i_k} \leq w_k$. Summing over these inequalities, we have $0 \leq \sum_{i=1}^{k} w_i$. This is

[1] Any possible inconsistency is caught in higher level routines of a refinement planner.

ALGORITHM: UPPER-ENVELOPE-AT-T
INPUT: An instance of the resource-envelope problem, and a time instant t.
OUTPUT: $g(t)$ and a flexible consistent schedule s that achieves it.

1. Construct the distance graph $\mathcal{D}(\mathcal{G})$ on the nodes of $\mathcal{G}$ as follows:
   a. For every edge $e = \langle X_i, X_j \rangle$ in $\mathcal{E}$:
      i. Add the edge $\langle X_i, X_j \rangle$ annotated with $UB(e)$.
      ii. Add the $\langle X_j, X_i \rangle$ annotated with $-LB(e)$.
2. For every $X_p \in \mathcal{P}$ and $X_c \in C$:
   a. Compute the shortest distance from X_p to X_c in $\mathcal{D}(\mathcal{G})$ (denoted $dist(X_p, X_c)$).
   b. Construct a (directed) size-2 conflict between X_p and X_c (denoted $X_c \rightarrow X_p$) if and only if $dist(X_p, X_c) < 0$.
3. Build a directed graph $E(\mathcal{G})$ as follows:
   a. The nodes of $E(\mathcal{G})$ correspond to events in $\mathcal{P} \cup \mathcal{C}$.
   b. The weight on X_i is set to $|r(X_i)|$.
   c. A directed edge $\langle X_p, X_c \rangle$ in $E(\mathcal{G})$ encodes a size-2 conflict $X_c \rightarrow X_p$.
4. Construct a graph $M(\mathcal{G})$ from $E(\mathcal{G})$ as follows:
   a. Remove a production node $X_p \in \mathcal{P}$ and all its adjacent edges if and only if $t + dist(X_p, X_0) < 0$.
   b. Remove a consumption node $X_c \in \mathcal{C}$ and all its adjacent edges if and only if $dist(X_0, X_c) - t < 0$.
5. Compute $Q = \{u_1, u_2 \ldots u_k\}$ as the largest weighted independent set in $M(\mathcal{G})$.
6. RETURN:
   a. $g(t) = \sum_{y_i \in \mathcal{P}} |r(y_i)|(y_i \in Q) - \sum_{y_i \in \mathcal{C}} |r(y_i)|(y_i \notin Q)$.
   b. $s = \mathcal{D}(\mathcal{G}) \cup \{\langle X_0, u_i \rangle$ annotated with t if $u_i \in \mathcal{P} \cap Q\} \cup \{\langle u_i, X_0 \rangle$ annotated with $-t$ if $u_i \in \mathcal{C} \cap Q\}$.

END ALGORITHM

Fig. 1. Shows the computation of $g(t)$ for a given time instant t, along with a flexible consistent schedule s that achieves it.

ALGORITHM: UPPER-ENVELOPE-ALL-T
INPUT: An instance of the resource-envelope problem.
OUTPUT: $g(t)$ for all t.

1. For all $X_p \in \mathcal{P}$:
   a. Insert $-dist(X_p, X_0)$ into list L.
2. For all $X_c \in \mathcal{C}$:
   a. Insert $+dist(X_0, X_c)$ into list L.
3. Sort L in ascending order $\langle d_1, d_2 \ldots d_{|L|} \rangle$.
4. For $i = 1, 2 \ldots |L| - 1$:
   a. Compute $g(d_i) =$ UPPER-ENVELOPE-AT-T at time d_i.
   b. Set $g(t) = g(d_i)$ for all t in the interval $[d_i, d_{i+1})$.
5. Set $g(t) = g(d_{|L|})$ for t in the interval $[d_{|L|}, +\infty)$.
6. Set $g(t) = 0$ in the interval $(-\infty, d_1)$.

END ALGORITHM

Fig. 2. Shows the computation of $g(t)$ for all t.

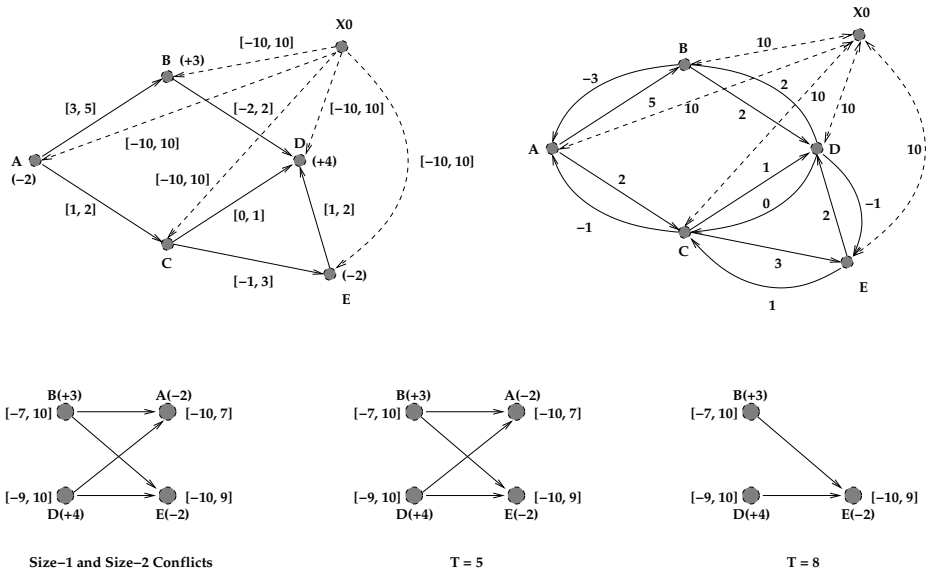

Fig. 3. Shows an instance of the resource-envelope problem cast as a bipartite matching problem. The top left diagram shows the original problem where nodes are annotated with their levels of production/consumption. The top right diagram shows the distance graph, and the bottom left diagram shows an encoding of the size-2 conflicts as a bipartite graph (the size-1 conflicts are represented using intervals annotating the nodes). The two other diagrams illustrate the change in the set of size-1 conflicts with t.

false since $\sum_{i=1}^{k} w_i$ is known to be negative. This means that there cannot exist any consistent schedule for $X_{i_1}, X_{i_2} \ldots X_{i_k}$. Conversely, if there are no negative cycles in $\mathcal{D}(\mathcal{G})$, then a consistent schedule exists for $X_1, X_2 \ldots X_n$, one of which is given by $X_i = d_{0i}$ where d_{0i} is the shortest distance from X_0 to X_i in $\mathcal{D}(\mathcal{G})$. Such a schedule satisfies all constraints in $\mathcal{D}(\mathcal{G})$. An edge $X_j - X_i \leq w_{ij}$ in $\mathcal{D}(\mathcal{G})$ is satisfied by $X_j = d_{0j}$ and $X_i = d_{0i}$ because $d_{0j} \leq w_{ij} + d_{0i}$ (since $w_{ij} + d_{0i}$ accounts for only one of the paths from X_0 to X_j).

Definition 1: A production event $p \in \mathcal{P}$ can contribute $+|r(p)|$ to the total production at time t, or not contribute at all. A consumption event $c \in \mathcal{C}$ can contribute $-|r(c)|$ to the total production at time t, or not contribute at all. An event is said to be *p-active* if it contributes its maximum to the total production by time t. This means that a production event p can be made *p-active* at time t if it is scheduled before (or at) t and a consumption event c can be made *p-active* at time t if it is scheduled after t.

Lemma 2: A schedule that achieves $g(t)$ at time t is also the one that maximizes $\sum_{y \in \mathcal{P} \cup \mathcal{C}} |r(y)| p\text{-}active(y)$.

Proof: We know that for any schedule s, $P_s(t) - C_s(t) = \sum_{p \in \mathcal{P}} |r(p)|(s(p) \leq t) - \sum_{c \in \mathcal{C}} |r(c)|(s(c) \leq t)$. Maximizing this is the same as maximizing $\sum_{c \in \mathcal{C}} |r(c)| + \sum_{p \in \mathcal{P}} |r(p)|(s(p) \leq t) - \sum_{c \in \mathcal{C}} |r(c)|(s(c) \leq t)$ because the additional term is inde-

pendent of s. Combining the first and the third terms, we have $\sum_{p\in\mathcal{P}} |r(p)|(s(p) \leq t) + \sum_{c\in\mathcal{C}} |r(c)|(s(c) > t)$. The last two terms yield $\sum_{y\in\mathcal{P}\cup\mathcal{C}} |r(y)|p\text{-}active(y)$ as required.

Lemma 3: A production event $p \in \mathcal{P}$ can be made *p-active* at time t if the addition of the edge $\langle X_0, p\rangle$ annotated with t does not result in a negative cycle in $\mathcal{D}(\mathcal{G})$. A consumption event $c \in \mathcal{C}$ can be made *p-active* at time t if the addition of the edge $\langle c, X_0\rangle$ annotated with $-t$ does not result in a negative cycle in $\mathcal{D}(\mathcal{G})$.

Proof: By definition, a production event p can be made *p-active* at time t if it is possible to schedule it before t. Retaining the semantics of the distance graph—that a constraint $X_b - X_a \leq w$ is specified as the edge $\langle X_a, X_b\rangle$ annotated with w—this corresponds to the addition of the edge $\langle X_0, p\rangle$ annotated with t to the distance graph without causing an inconsistency. Similarly, a consumption event c can be made *p-active* at time t if it is possible to schedule it after t, and this corresponds to the addition of the edge $\langle X_0, X_p\rangle$ annotated with $-t$ without causing an inconsistency. The truth of the Lemma then follows from the fact that inconsistencies correspond to negative cycles as stated in Lemma 1.

Definition 2: A *conflict* is a set of events all of which cannot be made *p-active* simultaneously at a given time t. A *minimal conflict* is a conflict no proper subset of which is also a conflict.

Lemma 4: A set of events can be simultaneously made *p-active* at time t if there is no subset of them that constitutes a minimal conflict.

Proof: By definition of a conflict, a set of events can be simultaneously made *p-active* at time t, if and only if there is no subset of them that constitutes a conflict. Further, the truth of the Lemma follows from the fact that a set of events constitutes a conflict if and only if some subset of them constitutes a minimal conflict.

Lemma 5: The size of a minimal conflict is ≤ 2.

Proof: A set of production events $\{p_1, p_2 \ldots p_k\}$ and a set of consumption events $\{c_1, c_2 \ldots c_l\}$ can be attempted to be made simultaneously *p-active* at time t if for all p_i, we can add the edge $\langle X_0, p_i\rangle$ annotated with t, and for all c_j, we can add the edge $\langle X_0, c_j\rangle$ annotated with $-t$, to the distance graph $\mathcal{D}(\mathcal{G})$ without creating a negative cycle. Let these edges be referred to as "special" edges and let $\mathcal{D}'(\mathcal{G})$ refer to the resulting distance graph. Knowing that $\mathcal{D}(\mathcal{G})$ does not have any negative cycles (because $\mathcal{E}$ is consistent), a negative cycle can occur in $\mathcal{D}'(\mathcal{G})$ only if it involves a "special" edge. Since all "special" edges have X_0 as an end point, a negative cycle must involve X_0. Further, since a fundamental cycle can have any node repeated at most once, at most 2 "special" edges can be present in a negative cycle in $\mathcal{D}'(\mathcal{G})$. Finally, since special edges correspond to *p-activation* of events, the size of a minimal conflict is ≤ 2.

Lemma 6: A size-2 conflict is independent of t.

Proof: Continuing the arguments in the proof of Lemma 5, when the size of a minimal conflict is 2, the negative cycle in $\mathcal{D}'(\mathcal{G})$ must involve an "incoming" "special" edge to X_0 (say $\langle c_j, X_0\rangle$) with weight $-t$ and an "outgoing" "special" edge from X_0 (say $\langle X_0, p_i\rangle$) with weight t. The weight of the negative cycle con-

taining exactly these two "special" edges is therefore $dist(p_i, c_j) + t - t$. This is independent of t and is negative if and only if $dist(p_i, c_j)$ is negative ($dist(p_i, c_j)$ is the shortest distance from p_i to c_j in $\mathcal{D}(\mathcal{G})$).

Lemma 7: A production event $p \in \mathcal{P}$ constitutes a size-1 conflict at time t when $t + dist(p, X_0) < 0$, and a consumption event $c \in \mathcal{C}$ constitutes a size-1 conflict at time t when $dist(X_0, c) - t < 0$.

Proof: Continuing the arguments in the proof of Lemma 5, when the size of a minimal conflict is 1 and it involves an "outgoing" "special" edge from X_0, it must be of the form $\langle X_0, p_i \rangle$ ($p_i \in \mathcal{P}$) with $dist(p_i, X_0) + t < 0$ (indicating that p_i cannot be *p-active* at time t). In the case that the minimal conflict involves an "incoming" "special" edge to X_0, it must be of the form $\langle c_j, X_0 \rangle$ ($c_j \in C$) with $dist(X_0, c_j) - t < 0$ (indicating that c_j cannot be *p-active* at time t).

Lemma 8: $g(t)$ and s are as computed in step 6 of Figure 1.

Proof: $M(\mathcal{G})$ incorporates the deletion of all size-1 conflicts and the computation of an independent set incorporates the absence of all size-2 conflicts. The computation of the largest weighted independent set in $M(\mathcal{G})$ therefore takes care of all minimal conflicts and targets the maximum possible *p-activity* at time t. Step 6(a) measures the total production corresponding to this *p-activity* and the required flexible consistent schedule s corresponds to the addition of edges required to *p-activate* the qualifying events.

Lemma 9: $g(t)$ is piecewise constant and changes only at a polynomial number of time points.

Proof: By Lemma 8, $g(t)$ is the largest weighted independent set in the graph $M(\mathcal{G})$. Since $M(\mathcal{G})$ is computed from $E(\mathcal{G})$ by deleting all size-1 conflicts at time t, the number of times $g(t)$ changes is equal to the number of times the set of size-1 conflicts changes in $M(\mathcal{G})$. Since $dist(X_0, c)$ ($c \in \mathcal{C}$) and $-dist(p, X_0)$ ($p \in \mathcal{P}$) respectively mark the membership of a consumption event c and a production event p in this set, the potential number of transition points for the piecewise constant function $g(t)$ is $O(|\mathcal{P}| + |\mathcal{C}|)$.

The complexity of the steps in Figure 1 that are independent of t (and can therefore be done just once) is dominated by the computation of shortest paths in the presence of negative cost edges using the Bellman-Ford algorithm. This is equal to $O(|\mathcal{X}||\mathcal{E}||\mathcal{P}||\mathcal{C}|)$. The time-dependent complexity is dominated by the computation of *maxflow* in a bipartite graph and is equal to $O((|\mathcal{P}|+|\mathcal{C}|)^{2.5})$. The complexity of the algorithm in Figure 2 is therefore equal to $O(|\mathcal{X}||\mathcal{E}||\mathcal{P}||\mathcal{C}| + (|\mathcal{P}| + |\mathcal{C}|)^{3.5})$.

3 Maximum Weighted Independent Set Computation

In this section, we present a polynomial-time algorithm for the computation of the largest weighted independent set in a bipartite graph (see Figure 4). A series of Lemmas are presented that prove the correctness of the algorithm. To keep the proofs of these Lemmas simple, we first deal with the case when all nodes have unit weight (imagine setting $|r(p_i)|$ and $|r(c_i)|$ to 1 in steps 1(b) and 1(c) of Figure 4). We then provide a single concluding Lemma that generalizes the

ALGORITHM: MAX-WT-IND-SET
INPUT: A bipartite graph $B = \langle P, C, E \rangle$ with $p_i \in P$ having weight $|r(p_i)|$ and $c_i \in C$ having weight $|r(c_i)|$.
OUTPUT: Maximum weighted independent set I^* in B.

1. Construct a directed graph $D = \langle U, Y \rangle$ as follows:
   a. $U = P \cup C \cup \{S, T\}$.
   b. For all $p_i \in P$: Y contains the edge $S \rightarrow p_i$ with capacity $|r(p_i)|$.
   c. For all $c_j \in C$: Y contains the edge $c_j \rightarrow T$ with capacity $|r(c_j)|$.
   d. For all $\langle p_i, c_j \rangle \in E$: Y contains the edge $p_i \rightarrow c_j$ of infinite capacity.
2. Compute a *maxflow* F (with residual graph R_F) in D from S to T.
3. Compute $H = \{\langle S, p_i \rangle | p_i \in P$ and p_i is unreachable from S in $R_F\} \cup \{\langle c_i, T \rangle |$ $c_i \in C$ and c_i is reachable from S in $R_F\}$.
4. Compute $V = \{p_i | \langle S, p_i \rangle \in H\} \cup \{c_i | \langle c_i, T \rangle \in H\}$.
5. RETURN: $I^* = P \cup C \backslash V$.

END ALGORITHM

Fig. 4. Shows the algorithm for computing the maximum weighted independent set in a bipartite graph.

correctness of the algorithm to the weighted version as required. We make use of the standard result that when all edges have integral capacities in an instance of the *maxflow* problem, a *maxflow* with integral amount of flow on all edges can be efficiently computed (see [1]).

Definition 3: A *matching* M in a bipartite graph B is a set of edges that do not share a common end-point. The *size of a matching* (denoted $|M|$) is the number of edges in it, and a *maximum matching* (denoted M^*) is a matching of maximum size. A *vertex cover* V in B is a set of nodes such that at least one end point of every edge is in it. A *minimum vertex cover* is one such that the total weight on all the nodes is minimized.

Lemma 11: If M^* is the maximum matching in B, then $|M^*| = F$.

Proof: For an integral flow, there cannot exist two edges of the form $\langle p_i, c_{j_1} \rangle$ and $\langle p_i, c_{j_2} \rangle$ both with non-zero flows. This is because the edge $\langle S, p_i \rangle$ has unit capacity and the flow has to be conserved at p_i. Similarly, there cannot exist two edges of the form $\langle p_{i_1}, c_j \rangle$ and $\langle p_{i_2}, c_j \rangle$ both with non-zero flows because $\langle c_j, T \rangle$ is of unit capacity. Therefore, an integral flow in D defines a matching in B of the same size and a *maxflow* F in D defines a maximum matching M^* in B of the same size, hence making $|M^*| = F$.

Lemma 12: If V^* is the minimum vertex cover in B, then $|V^*| \geq |M^*|$.

Proof: For any edge in M^*, at least one of its end points must be in V^*. Also, since no two edges in M^* share a common end point, they cannot be covered by the same element in V^*. This means that $|V^*| \geq |M^*|$.

Lemma 13: $|V| = F$ (F is the *maxflow* in D and V is the vertex cover for B constructed in step 4 of Figure 4).

Proof: From the construction of V, $p_i \in P$ is in V if and only if $\langle S, p_i \rangle$ is in H and $c_j \in C$ is in V if and only if $\langle c_j, T \rangle$ is in H. This means that $|V| = |H|$. Since H is formed out of considering all edges that have one end reachable from

S and the other unreachable in R, it constitutes a *minimum cut* between S and T in D. From the *maxflow-mincut* Theorem, $|H| = F$, and hence $|V| = F$ as required.

Lemma 14: For the bipartite graph B, if V^* is the minimum vertex cover, then $|V| = |V^*|$ (where V is the vertex cover constructed for B in step 4 of Figure 4).

Proof: From the above Lemmas, we have that $|V| = F$, $|V^*| \geq |M^*|$ and $|M^*| = F$. This means that $|V^*| \geq |V|$. Since V^* is the optimal vertex cover by definition, we have that $|V| = |V^*|$ and is the required minimum vertex cover.

Lemma 15: If I^* is the largest independent set, then $|I^*| + |V^*| = |P \cup C|$. Moreover, $P \cup C \backslash V$ is an optimal independent set.

Proof: Consider any vertex cover U. $P \cup C \backslash U$ does not contain any two nodes of the form $p_i \in P$ and $c_j \in C$ such that there is an edge $\langle p_i, c_j \rangle$ between them. This means that $P \cup C \backslash U$ is an independent set. Also, since $P \cup C \backslash U$ and U form a partition of $P \cup C$, we have that $|P \cup C \backslash U| + |U| = |P \cup C|$. When U is the minimum vertex cover V^* we have that $|I^*| + |V^*| = |P \cup C|$. Finally since V computed in step 4 of Figure 4 is optimal (Lemma 14), we have that $P \cup C \backslash V$ is the required largest independent set in B.

Lemma 16: The algorithm presented in Figure 4 works for arbitrary positive weights $|r(y_i)| > 0$.

Proof: From the foregoing Lemmas, we know that the algorithm works for unit weights—i.e. $|r(y_i)| = 1$ for $y_i \in P \cup C$. Now suppose that the weights were positive integers (still not the general case). Conceptually, a new bipartite graph can be constructed where node y_i (in $P \cup C$) with weight $|r(y_i)|$ is replicated $|r(y_i)|$ times—each of unit weight and independent of each other. An edge $p_i \rightarrow c_j$ entails all copies of p_i (denoted $p'_{i_1}, p'_{i_2} \ldots p'_{i_{|r(p_i)|}}$) to have an edge to all copies of c_j (denoted $c'_{j_1}, c'_{j_2} \ldots c'_{j_{|r(c_j)|}}$). The staged *maxflow* in D will then have all copies of y_i behaving identically. Also since all edges of the form $\langle p'_{i_k}, c'_{j_l} \rangle$ have infinite capacity, we can replace the group of edges $\langle S, p'_{i_1} \rangle, \langle S, p'_{i_2} \rangle \ldots \langle S, p'_{i_{|r(y_i)|}} \rangle$ (each of unit capacity) with a single edge $\langle S, p_i \rangle$ of capacity $|r(p_i)|$, and similarly replace all edges of the form $\langle c'_{j_1}, T \rangle, \langle c'_{j_2}, T \rangle \ldots \langle c'_{j_{|r(c_j)|}}, T \rangle$ (each of unit capacity) with a single edge $\langle c_j, T \rangle$ of capacity $|r(c_j)|$. All intermediate edges are of infinite capacity and are defined (as previously) using the idea of directed size-2 conflicts. Now consider the most general case where $|r(y_i)|$ is positive but need not be an integer. In such a case, the idea is to conceptually scale all the weights by a uniform factor L to convert all of them to integers. We can then find the largest independent set using the scaled weights and since uniform scaling does not affect the largest weighted independent set, the same computed set can then be used after scaling down the weights by L. Computationally however, the idea of scaling is not reflected anywhere except in the fact that the weights $r(y_i)$ can be used as they are to define capacities on the edges in D.

4 Incremental Computation

A refinement planner proceeds by refining and extending partial plans. The refinement operators used to extend partial plans include the addition of new

actions to satisfy open conditions or sub-goals, and the addition of new temporal constraints to resolve threats between actions (see [6]). Because the resource-envelopes need to be computed at each stage to guide the search for a good plan, incremental computation becomes extremely important. In general, there are two places where incremental computation can be leveraged: (1) in the computation of the envelopes across points of discontinuity (for the same set of events and constraints), and (2) in the computation of the envelopes as and when new events and constraints (reflecting the refinement of partial plans) are added. The second case is more general and we deal with it directly.

4.1 Incremental *maxflow* in Bipartite Graphs

Because the computation of the envelopes involves computing the *maxflow* in bipartite graphs, we will first show how to make this incremental—i.e., we will show how we can reuse the computation for one instance of the *maxflow* problem on bipartite graphs into solving another instance. The complexity of this incremental algorithm is analyzed in terms of the parameters that characterize the difference between the two instances.

We will denote a staged *maxflow* problem in bipartite graphs by $\langle P, C, E, S, T\rangle$. Here, P is the set of production events, C is the set of consumption events, E is the set of edges between P and C (each of them assumed to be of infinite capacity as is indeed so in the context of computing envelopes), and S and T are respectively the source and terminal nodes. The incremental *maxflow* problem is then to solve $\langle P, C, E, S, T\rangle$ given the solution for $\langle P', C', E', S, T\rangle$.

Figure 5 shows the incremental computation of *maxflow* for $\langle P, C, E, S, T\rangle$ using the residual graph carried over from the computation of *maxflow* for $\langle P', C', E', S, T\rangle$. Figure 6 illustrates the working of this algorithm on a small example. Central to the algorithm is the exploitation of the fact that the complexity of solving a *maxflow* problem can be characterized both by the topology of the graph (like it is easier in bipartite graphs than in general) and the value of the *maxflow* itself. The algorithm makes use of calls to *maxflow* on morphed instances of the problem (steps 6 and 9 in Figure 5) which are assumed to be solved directly using greedy flow augmentation methods within a complexity of $O(m|f^*|)$ ($|f^*|$ is the value of the *maxflow* and m is the number of edges in the graph) [1]. These morphed instances are assured of having a "small" $|f^*|$, hence making the complexity of the algorithm much better than re-computation from scratch. A series of Lemmas are presented that establish the correctness of the algorithm.

Lemma 17: A feasible flow F for $\langle P_{new}, C_{new}, D_{new}, S, T\rangle$ is also feasible for $\langle P, C, E, S, T\rangle$ if its associated residual graph R_F has residual capacity 0 on all edges of the form $\{\langle u, v\rangle | u \in C'\backslash C \text{ or } v \in P'\backslash P\}$.

Proof: A feasible flow for $\langle P_{new}, C_{new}, D_{new}, S, T\rangle$ is also so for $\langle P, C, E, S, T\rangle$ if no flow is pushed through any of the edges in D_{new} that are not present in $\langle P, C, E, S, T\rangle$. If there were such an edge, then it must have been of the form $\langle u, v\rangle$ with $u \in P'\backslash P$ or $v \in C'\backslash C$, and its utilization would create a positive residual capacity in the opposite direction—viz. in $\langle v, u\rangle$. The truth of the

ALGORITHM: INCR-MAX-FLOW
INPUT: $\langle R', f', M' \rangle$ for $\langle P', C', E', S, T \rangle$, and a new instance $\langle P, C, E, S, T \rangle$.
OUTPUT: $\langle R, f, M \rangle$ for $\langle P, C, E, S, T \rangle$.

01. Create a bipartite graph $\langle P_{new}, C_{new}, D_{new} \rangle$ such that:
  a. $P_{new} = P \cup P'$.
  b. $C_{new} = C \cup C'$.
  c. $D_{new} = \{\langle S, u \rangle$ with capacity $|r(u)|$ s.t. $u \in P \backslash P'\} \cup \{\langle v, T \rangle$ with capacity $|r(v)|$ s.t. $v \in C \backslash C'\} \cup \{\langle u, v \rangle$ with infinite capacity s.t. $u \in P \backslash P'$ or $v \in C \backslash C'$ and $\langle u, v \rangle \in E\} \cup \{\langle u, v \rangle$ with capacity $R'(\langle u, v \rangle) | \langle u, v \rangle$ is an edge in $R'\}$.
02. For all edges $\langle u, v \rangle$ in D_{new} in the set $\{\langle u, v \rangle$ s.t. $u \in C' \backslash C$ or $v \in P' \backslash P\}$:
  a. Set $f_{back}(\langle u, v \rangle) = R'(\langle u, v \rangle)$.
  b. Set $R_{back}(\langle u, v \rangle)$ and $R_{back}(\langle v, u \rangle)$ to 0.
03. For all edges $\langle u, v \rangle$ in D_{new} not in the set $\{\langle u, v \rangle, \langle v, u \rangle$ s.t. $u \in C' \backslash C$ or $v \in P' \backslash P\}$:
  a. Set $R_{back}(\langle u, v \rangle) = R'(\langle u, v \rangle)$.
04. For all nodes u in $P_{new} \cup C_{new}$:
  a. Set $ex(u) = \sum_{\langle v,u \rangle \in D_{new}} f_{back}(\langle v, u \rangle) - \sum_{\langle u,v \rangle \in D_{new}} f_{back}(\langle u, v \rangle)$.
05. Create a directed graph G_{morph} from R_{back} as follows:
  a. Add new nodes S' and T'.
  b. For all nodes $u \in P_{new} \cup C_{new}$:
    i. If $ex(u) > 0$, add the edge $\langle S', u \rangle$ with capacity $ex(u)$.
    ii. If $ex(u) < 0$, add the edge $\langle u, T' \rangle$ with capacity $-ex(u)$.
  c. Add edges $\langle S, T \rangle$ and $\langle T, S \rangle$ of infinite capacities.
06. Solve for $\langle R_{cons}, f_{cons}, M_{cons} \rangle = maxflow$ on G_{morph} from S' to T'.
07. For all edges $e \in D_{new}$:
  a. Set $f_{legal}(e) = f_{cons}(e) + f_{back}(e)$.
08. Create a graph G_{final} from R_{cons} as follows:
  a. Remove S' and T'.
  b. Remove all edges of the form $\langle u, v \rangle$ where $\{u, v\} \cap \{S', T'\} \neq \{\}$.
  c. Remove the edges $\langle S, T \rangle$ and $\langle T, S \rangle$.
09. Solve for $\langle R_{final}, f_{final}, M_{final} \rangle = maxflow$ on G_{final} from S to T.
10. Build a graph R_{return} from R_{final} by removing all nodes u and their incident edges when $u \in P' \backslash P \cup C' \backslash C$.
11. For all edges $\langle u, v \rangle$ in $E \cup \{\langle S, u \rangle$ s.t. $u \in P\} \cup \{\langle v, T \rangle$ s.t. $v \in C\}$:
  a. Set $f_{return}(\langle u, v \rangle) = R_{return}(\langle v, u \rangle)$.
12. Set $M_{return} = \sum_{u \in P} f_{return}(\langle S, u \rangle)$.
13. RETURN: $\langle R_{return}, f_{return}, M_{return} \rangle$.

END ALGORITHM

Fig. 5. Shows the algorithm for incremental *maxflow* in bipartite graphs. The idea is to reverse the flow on edges that are not present in the new instance by first computing the amount by which such a reverse flow would violate the conservation constraints at intermediate nodes. After the excess at each node is measured, a *maxflow* is staged between two auxiliary nodes S' and T' to regain conservation consistency. A subsequent maximization phase is carried out that respects this consistency.

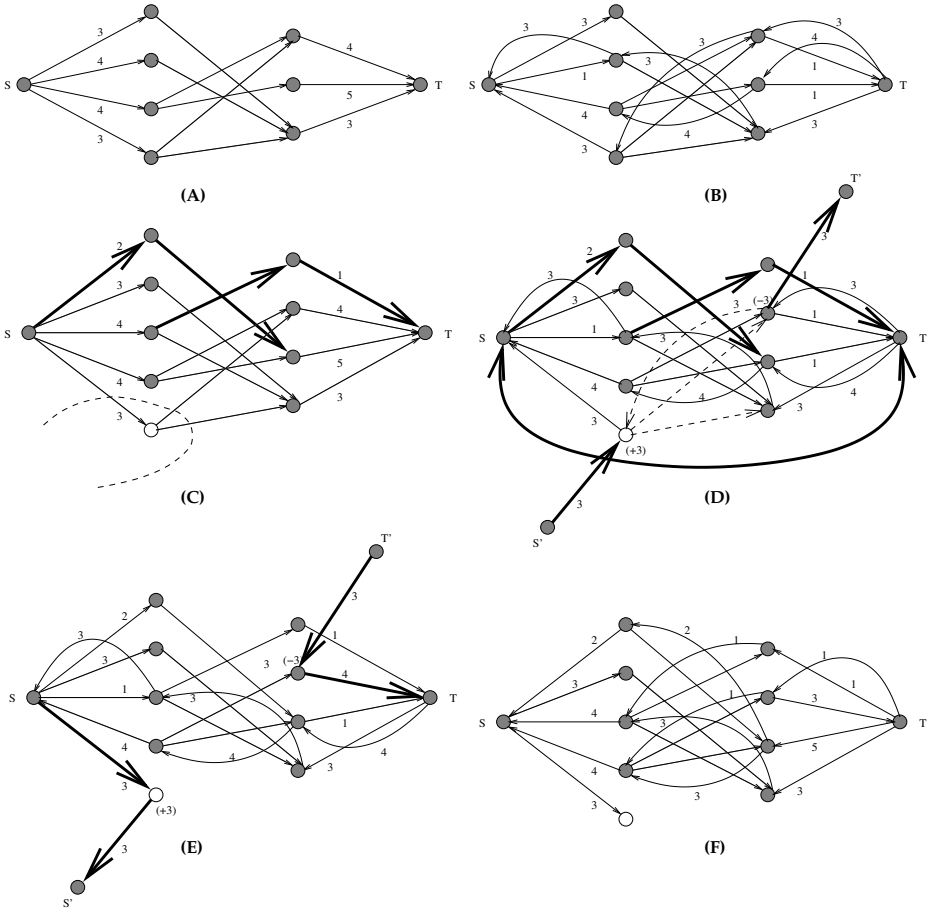

Fig. 6. Illustrates an example for the working of the algorithm in Figure 5 (all edges not explicitly annotated with their capacities are assumed to be of infinite capacities). (A) shows the staged *maxflow* on a bipartite graph with two extra nodes S and T, which are also the source and terminal nodes for the *maxflow* respectively. A *maxflow* computation on this problem results in a residual graph as shown in (B). (C) shows a slightly different problem compared to that in (A). The dark lines indicate the addition of extra elements and the dotted lines indicate deletions. The idea is to solve this instance by making use of the residual graph in (B). (D) illustrates the idea of reversing the flow on illegal edges by computing the excess at each node and deleting the illegal edges as indicated by the dotted lines. (E) indicates the result of a *maxflow* computation between S' and T' to regain feasibility. Finally, (F) shows the computation of a *maxflow* between S and T without pushing back any flow on the illegal edges, as required.

Lemma then follows from the contrapositive of this implication.

Lemma 18: f_{legal} (see step 7) is a feasible flow for $\langle P_{new}, C_{new}, D_{new}, S, T\rangle$.

Proof: A flow is feasible when (1) all capacity constraints are satisfied, and (2) at every node (except S and T) conservation constraints are preserved—i.e. the total incoming flow to that node = total outgoing flow from that node. Condition (1) holds because $f_{back}(e)$ is set to either $R'(e)$ or 0 (see step 2). In the former case, $R_{back}(e)$ is set to 0 and hence $f_{cons}(e)$ (which is a feasible flow for G_{morph} obtained from the residual graph R_{back}) is 0. Together, $f_{back}(e) + f_{cons}(e) \leq R'(e)$ as required. In the latter case, $R_{back}(e)$ is set to $R'(e)$ (see step 3) and $f_{cons}(e) + f_{back}(e) = f_{cons}(e)$ which we know is $\leq R_{back}(e) = R'(e)$ (from the feasibility of f_{cons} for G_{morph}) as required. Now consider condition (2). The flow f_{back} does not heed to conservation constraints, but we will prove that the projection of f_{cons} on all edges except those that involve S' or T', exactly compensates for this. Step 4(a) first accounts for the excess at every node—viz. net inflow – net outflow. Now suppose that f_{cons} saturated all edges of the form $\langle S', u\rangle$ and $\langle v, T'\rangle$. Then by construction, the net inflow (w.r.t. f_{cons}) to a node u with positive excess = net outflow = $ex(u)$. Now if we take the projection of f_{cons} on all edges except those that involve S' or T', we have that the net inflow = 0 and net outflow = $ex(u)$, exactly compensating for $f_{back}(e)$. A similar argument holds when $ex(u)$ is negative. It is easy to observe that in fact, all edges of the form $\langle S', u\rangle$ and $\langle v, T'\rangle$ are saturated after step 6. This is because the edges $\langle S, T\rangle$ and $\langle T, S\rangle$ are made to be of infinite capacities in step 5(c), and the consistency of R' for $\langle P', C', E', S, T\rangle$ ensures that $R'(\langle u, S\rangle) \geq |ex(u)|$ when $ex(u) > 0$ and $R'(\langle T, v\rangle) \geq |ex(v)|$ when $ex(v) < 0$.

Lemma 19: f_{legal} is a feasible flow for $\langle P, C, E, S, T\rangle$.

Proof: From the above two Lemmas, it suffices to prove that the associated residual graph of f_{legal} viz. R_{cons} has residual capacity 0 on all edges of the form $\{\langle u, v\rangle | u \in C'\backslash C$ or $v \in P'\backslash P\}$. Since all these edges and those of the form $\{\langle v, u\rangle | u \in C'\backslash C$ or $v \in P'\backslash P\}$ are made to have capacity 0 before computing f_{legal} (step 2(b)), the final residual capacity on these edges remains 0 as required.

Lemma 20: G_{final} is the residual graph for f_{legal} on $\langle P, C, E, S, T\rangle$ (see step 8 in Figure 5).

Proof: R_{back} is the residual graph for f_{back}. Since f_{cons} works on R_{back} (derived from G_{morph}) directly, the projection of R_{cons} on all edges that do not involve S' or T' (viz. G_{final}) is the residual graph for $f_{cons} + f_{back} = f_{legal}$.

Lemma 21: R_{return} is the required residual graph for $\langle P, C, E, S, T\rangle$.

Proof: From the previous Lemma, G_{final} is the residual graph for f_{legal}. f_{legal} is just a legal flow and is not necessarily the *maxflow*. Step 9 ensures that any further flow that can be augmented between S and T is indeed done so by maintaining the legality constraints of using a capacity of 0 for all edges of the form $\langle u, v\rangle$ or $\langle v, u\rangle$ with $u \in C'\backslash C$ or $v \in P'\backslash P$ (step 2(b)). R_{return} is then the required residual graph for $\langle P, C, E, S, T\rangle$ given that it is the projection of R_{final} on $\{S, T\} \cup P \cup C$ (the legal nodes in the new instance).

Lemma 22: f_{return} and M_{return} are the required maximum flow and its value correspondingly.

ALGORITHM: INCR-SHRT-PATHS
INPUT: D' and Π' for $\mathcal{G}' = \langle \mathcal{X}', \mathcal{E}', \mathcal{P}', \mathcal{C}' \rangle$, and a new instance $\mathcal{G} = \langle \mathcal{X}, \mathcal{E}, \mathcal{P}, \mathcal{C} \rangle$, with $\mathcal{X}' \subseteq \mathcal{X}$, $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{P}' \subseteq \mathcal{P}$, and $\mathcal{C}' \subseteq \mathcal{C}$.
RESULT: D and Π for $\mathcal{G}$.
1. For all $i \in \mathcal{X}$ and $p \in \mathcal{P} \cup \{X_0\}$:
   a. Set $d_{p,i} = d'_{p,i}$.
   b. Set $\pi_{p,i} = \pi'_{p,i}$.
2. For each edge $\langle u, v \rangle \in \mathcal{E} \backslash \mathcal{E}'$:
   a. RELAX-EDGE$(u, v, \mathcal{P}')$.
3. For each edge $\langle u, v \rangle \in \mathcal{E}$:
   a. RELAX-EDGE$(u, v, \mathcal{P} \backslash \mathcal{P}')$.
END ALGORITHM

ALGORITHM: RELAX-EDGE
INPUT: edge $\langle u, v \rangle$ and a set of production events P.
RESULT: modifications in D and Π.
1. If $P = \{\}$ RETURN.
2. For all $p \in P$:
   a. If $d_{v,p} > d_{u,p} + UB(\langle u, v \rangle)$:
      i. $d_{v,p} = d_{u,p} + UB(\langle u, v \rangle)$.
      ii. $\pi_{v,p} = u$.
   b. Else:
      i. Remove p from P.
3. For all edges $\langle v, y \rangle$:
   a. RELAX-EDGE(v, y, P).
END ALGORITHM

Fig. 7. Shows the incremental computation of shortest paths required for posing new bipartite matching problems as and when search proceeds. $d_{p,i}$ is the current best estimate of the shortest path from the production event p to the node i, and $\pi_{p,i}$ is the predecessor node of i in the shortest path from p to i. D and Π are the corresponding 2D arrays.

Proof: Given that R_{return} is the required residual graph, the flow on any edge $\langle u, v \rangle$ is given by the residual capacity in the opposite direction and is computed in step 11. Similarly, M_{return} is the sum of the flows on all edges outgoing from S, and is computed in step 12.

The complexity of the incremental *maxflow* algorithm in Figure 5 is dominated by steps 6 and 9. Step 6 has a complexity of $O(m|f_{cons}|)$ and step 9 has a complexity of $O(m|f_{final}|)$. By construction, $|f_{cons}| \leq \sum_{\{y_i \in P' \backslash P \cup C' \backslash C\}} |r(y_i)|$ and $|f_{final}| \leq \sum_{\{y_i \in P \backslash P' \cup C \backslash C'\}} |r(y_i)|$. When the computation of the resource envelopes needs to be done frequently, the sets $P \backslash P'$, $P' \backslash P$, $C \backslash C'$ and $C' \backslash C$ are very small, and the complexity of the algorithm is only about $O(m)$. This is significantly lesser than the complexity of recomputing the envelopes from scratch—viz. $O(n^{2.5})$. The total amount of work that we save is therefore equal to this difference (at each step) times the size of the search space (which is usually exponential).

4.2 Incremental Shortest Path Computation

All other incremental computation required to be done to pose a new bipartite *maxflow* problem as and when search proceeds by making partial commitments, can be done using algorithms similar to the Bellman-Ford algorithm for computing shortest path distances. Figure 7 illustrates this computation. Because of its similarity to the standard Bellman-Ford algorithm (see [1]), we do not provide a rigorous proof of its correctness in this paper. At any stage, the set of size-1 conflicts is given by all $p \in \mathcal{P}$ such that $d_{p,X_0} < -t$, and all $c \in \mathcal{C}$ such that $d_{X_0,c} < t$. The set of size-2 conflicts is given by all $c \rightarrow p$ ($c \in \mathcal{C}$ and $p \in \mathcal{P}$) such that $d_{c,p} < 0$.

The complexity of the incremental shortest path computation is $O(|\mathcal{E}\backslash\mathcal{E}'||\mathcal{E}||\mathcal{P}|+|\mathcal{P}\backslash\mathcal{P}'||\mathcal{E}|^2)$. When the computation of the resource envelopes needs to be done frequently, the sets $\mathcal{P}\backslash\mathcal{P}'$ and $\mathcal{E}\backslash\mathcal{E}'$ are very small, and the complexity of algorithm is only about $O(|\mathcal{E}|^2)$. This is significantly lesser than the complexity of recomputing the shortest path distances from scratch—viz. $O(|\mathcal{P}||\mathcal{E}|^2)$. The total amount of work that we save is therefore equal to this difference (at each step) times the size of the search space (which is usually exponential).

5 Conclusions and Future Work

We described efficient algorithms for the incremental computation of resource envelopes in producer-consumer models. This is important in all contexts where the computation of resource envelopes can potentially be used to guide the search for a good plan. In the context of interleaved planning and scheduling, a refinement planner proceeds by making partial commitments and the resource envelopes need to be recomputed at each point in the search space. In the context of interleaved planning and execution monitoring, execution of a plan may not always result in the intended outcome and fast re-planning is necessary. Incremental computation of resource envelopes then becomes extremely important for an active management of planning and execution monitoring. The algorithms presented in this paper are also constructive in that they yield flexible consistent schedules that actually achieve $g(t)$ or $h(t)$ for any given time point. This ensures both good performance at bottle-neck points (as argued in [3]) and robustness with respect to exogenous events and uncertainty of execution.

We are also currently working on interesting approximation algorithms for optimal plan scheduling that use the resource envelope computation as an important subroutine. Future work will also pursue empirical verification of using resource envelopes within the general framework of interleaved planning and scheduling, and interleaved planning and execution monitoring.

References

1. Cormen T. H., Leiserson, C. E. and Rivest, R. L. 1990. Introduction to Algorithms. *Cambridge, MA, 1990.*
2. Laborie P. 2001. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. *ECP 2001.*
3. Muscettola N. 1994. On the Utility of Bottleneck Reasoning for Scheduling. *AAAI 1994.*
4. Muscettola N. 2002. Computing the Envelope for Stepwise-Constant Resource Allocations. *CP 2002.*
5. Smith D., Frank J. and Jonsson A. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review 15:1, 2000.*
6. Nguyen X. and Kambhampati S. 2001. Reviving Partial Order Planning. *IJCAI-2001.*