

Bounding the Resource Availability of Partially Ordered Events with Constant Resource Impact

Jeremy Frank

Computational Sciences Division
NASA Ames Research Center, MS 269-3
frank@email.arc.nasa.gov
Moffett Field, CA 94035

Abstract. We describe a resource bounding technique called the *Flow Balance Constraint* (*FBC*) to tightly bound the amount of available resource for a set of partially ordered events with piecewise constant resource impact. We compare this technique with two existing techniques, the Balance Constraint (*BC*) due to Laborie and the Resource Envelope (E_t) due to Muscettola. We prove that *FBC* generates smaller proof trees than either *BC* or E_t under chronological search with a static variable and value order. We also show that E_t and *BC* are not strictly comparable in terms of the size of the search trees generated under chronological search with a static variable and value order. We provide an efficient algorithm for calculating *FBC* and bound its complexity. We then show how to generalize *FBC* to construct tighter resource bounds but at increased computational cost.

1 Introduction

Scheduling requires ordering tasks while simultaneously satisfying temporal constraints and resource availability constraints. Following Muscettola [1], Laborie [2] has provided a simple but expressive formalism for some scheduling problems called *Resource Temporal Networks* (RTNs) that we will study in this paper. Briefly, RTNs consist of a *Simple Temporal Network* (STN) as described in [3], constant resource impacts (either production or consumption) for events, and piecewise constant resource bounds. Usually, scheduling is performed assuming that the problem's characteristics are known in advance, do not change, and that the execution of the schedule is deterministic. Often, these assumptions are violated in practice. For example, if events do not take place exactly when they are scheduled, it may be costly to find a new schedule. Maintaining *temporal flexibility* during scheduling (as was done in [4]) permits the construction of a schedule without determining exactly when events take place until execution. Building schedules by ordering events rather than assigning event times preserves temporal flexibility. Techniques such as that described in [5] make it possible to efficiently update the flexible schedule once the precise timing of events are known.

It is straightforward to calculate the resource utilization over time of events whose execution time are fixed. The task becomes more difficult when activities or events are not fixed in time, and more difficult still when events can have arbitrary impact on resources. An important use of these techniques is to provide a means for halting

the scheduling process, either by determining that the resource constraint is satisfied or proving that it can't be satisfied, implying that some scheduling decisions need to be changed. In the context of constructive search, early detection of success or failure is often important to achieving good search performance. In this paper we focus on the ability of resource bounding techniques to reduce the cost of constructive search algorithms by means of detecting success or failure.

Two existing techniques address the bounding of resource availability for activities or events with a constant resource impact, the Balance Constraint (BC) [6] due to Laborie, and the Resource Envelope (E_t) [1] due to Muscettola. These techniques are described in more detail in Section 2. The techniques are somewhat different, with BC featuring an efficient (but loosely) bounding approximation, while E_t is somewhat more costly but provides a tight bound (in all cases a schedule justifying the bound is proved to exist). Somewhat surprisingly, these techniques are not strictly comparable in terms of the size of the search trees generated under chronological search. We provide examples demonstrating this in Section 3. In Section 4 we describe the *Flow Balance Constraint* (FBC), a novel synthesis of the approaches described in [6, 1]. In Section 5 we prove that FBC generates smaller proof trees under chronological search. In Section 6 we describe the complexity of a naive algorithm to calculate FBC . We then exploit the results of [7] to calculate FBC incrementally, thereby reducing its computational cost. In Section 7 we then generalize FBC in order to construct even tighter resource bounds but at increased computational cost. Finally, in Section 8 we conclude and describe future work.

2 Previous Work

In this paper we will assume that time is represented using integers¹. We will also assume that the resource impact of each event is known when solving begins. We will assume each RTN has only one resource, that there is one less-than constraint, and one greater-than 0 constraint on the resource. We also assume that the lower bound of the scheduling horizon is 0, and that the resource has its maximum availability at time 0. The results that follow do not depend on these assumptions.

The techniques we study are aimed at checking the *Necessary Truth Criterion* [2, 8] (NTC), namely, whether there exists a feasible solution of the STN that also satisfies the resource constraints. As with all constraints, for a given RTN the NTC can be either proved satisfied, proved unsatisfiable, or neither. The NTC is proved satisfied if the maximum calculated availability of the resource is always an upper bound on the actual maximum available resource and is always below the upper limit, and the minimum calculated availability of the resource is always a lower bound on the actual minimum available resource and is always above 0. The NTC is proved unsatisfied if the maximum calculated availability of the resource is always an upper bound on the actual maximum available resource and is ever below 0, or the minimum calculated availability of the resource is always a lower bound on the actual minimum available resource and is ever above the resource upper bound. Otherwise, the satisfiability of the NTC is undetermined.

We will use the following notation: Let \mathcal{N} be the set of all events of an RTN and $n = |\mathcal{N}|$. Let $X \in \mathcal{N}$; $c(X)$ denotes the resource impact of X . If $c(X) < 0$ X is said to be a *consumer*; if $c(X) > 0$ then X is said to be a *producer*. As defined in [1], X *anti-precedes* Y if X must occurs at or after Y (i.e. $Y \leq X$). A *predecessor set* of X is a set such that if $X \in S$ then every event Y such that $Y \leq X$ is also in S . A *successor set* of X is a set such that if $X \in T$ then every event Y such that $Y > X$ is also in T . Let R be an RTN and let $A(R)$ be any procedure for evaluating the NTC. If $A(R) = T$ then the NTC is provably satisfied. If $A(R) = F$, the NTC is provably unsatisfied. If $A(R) = ?$ the NTC is neither provably satisfied nor provably unsatisfied.

¹ This assumption can be relaxed, but leads to resource bounds holding over half-open intervals of time.

2.1 The Balance Constraint

Laborie's *Balance Constraint (BC)* [6] calculates bounds on the maximum and minimum amount of a resource immediately prior to and immediately following the execution of an event X . The STN is partitioned relative to X into the following sets: Before $B(X)$, Before or Equal $BS(X)$, Equal $S(X)$, Equal or After $AS(X)$, After $A(X)$, and Unordered $U(X)$. These sets are then used to calculate bounds on the following quantities: $L_{min}^<(X)$ for the minimum available resource before X happens, $L_{max}^<(X)$ for the maximum available resource before X happens, $L_{min}^>(X)$, for the minimum available resource after X has happened, and $L_{max}^>(X)$ for the maximum available resource after X has happened. A sample calculation: let $P_{max}^<(X) = (B(X) \cup (V \in BS(X) \cup U(X) | c(V) > 0))$ Then an upper bound on $L_{max}^<(X) = \sum_{Z \in P_{max}^<(X)} c(Z)$. The other bounds can be constructed in a similar manner. The bounds are loose in that no schedule consistent with the existing constraints may achieve the bound. The computational complexity of BC is dominated by the maintenance of arc-consistency of the STN.

2.2 The Resource Envelope

The *Envelope Algorithm* E_t [1] finds schedules permitted by the STN that maximize or minimize the available resource at a given time t . The *envelope* of an RTN is the pair of functions from time to the maximum ($L_{max}(t)$) and minimum ($L_{min}(t)$) available resource defined by the current STN. Events are partitioned into those that must have occurred by t (*Closed*) $C(t)$, are permitted to occur at t (*Pending*) $P(t)$, or can't occur until after t (*Open*) $O(t)$. The maximum available resource at a time t , $L_{max}(t)$, is supported by a schedule ensuring that the events in $P_{max}(t) \subset (P(t) \cup C(t))$ all occur before t . To find the set $\Delta P_{max}(t) \subset P(t)$ that contributes to $P_{max}(t)$, a maximum flow problem is constructed using all anti-precedence links derived from the arc-consistent STN. The rules for building the flow problem to find $L_{max}(t)$ are as follows: all events in $P(t)$ are represented by nodes of the flow problem. If $Y \geq X$ then the flow problem contains an arc $X \rightarrow Y$ with infinite capacity. If $c(X) > 0$ then the problem contains an arc $\sigma \rightarrow X$ with capacity $c(X)$. If $c(X) < 0$ then the problem contains an arc $Y \rightarrow \tau$ with capacity $|c(X)|$. The flow problem to find $L_{min}(t)$ is constructed in a similar manner. Examples of the flow problem construction are shown in Figure 1. The maximum flow of this flow network matches all possible production with all possible consumption in a manner consistent with the precedence constraints. The predecessor set of those events reachable in the residual flow is $\Delta P_{max}(t)$. Define $\Delta P_{max}^C(t) = P(t) - \Delta P_{max}(t)$. The tightness of the bound is guaranteed by proving that adding the constraints $\{X_{ub} \leq t\} \forall X \in \Delta P_{max}(t)$ and $\{Y_{ub} > t\} \forall Y \in \Delta P_{max}^C(t) \cup O(t)$ is consistent with the original STN.

It turns out that $\Delta P_{max}(t)$ need only be computed at $\leq 2n$ distinct times, which we refer to as *Instants*. The set of these times $I = \{X_{lb} \cup X_{ub}\} \forall X \in \mathcal{N}$ is the set of all lower and upper bounds of events. Obviously the set of events that could define $L_{max}(t)$ does not change between consecutive instants. The complexity of the algorithm described in [1] is $O(n * \text{MaxFlow}(n, m))$, where $n = |\mathcal{N}|$ and m is the number of anti-precedence relationships in the arc-consistent STN. In [7] this is reduced to $O(\text{MaxFlow}(n, m))$ by taking advantage of the order in which edges are added to and removed from a single maximum flow problem. The crucial observation is that when computing E_t an event always moves from open to pending to closed and stays closed. As shown in Figure 1, this guarantees that events topologically "late" in the flow problem are removed, while events topologically "early" are added to the flow problem. The insight is that the flow problem need not be solved anew, but the previous flow can be reused, thereby saving a factor of n . As an aside, a more general incremental

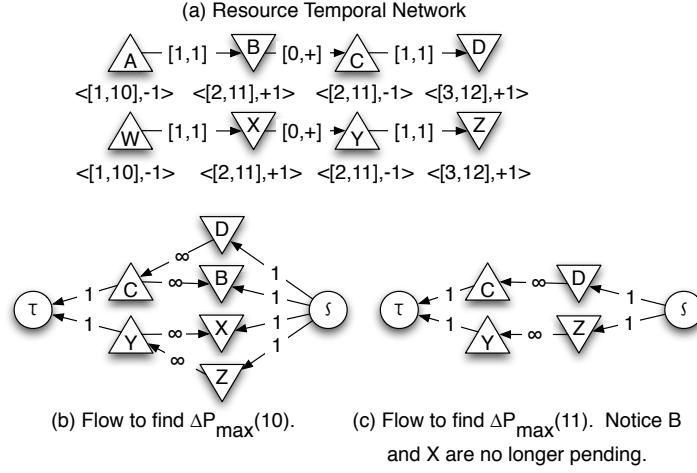


Fig. 1. The Flow(s) for E_t .

maximum flow algorithm is given in [9], but it doesn't take advantage of the order of flow arc additions and deletions, and is not as efficient at calculating the envelope.

3 Examples of Non-Domination

Muscettola previously demonstrated that E_t can prove the NTC is satisfied in cases where BC cannot. In this section we provide an example where E_t fails to prove the NTC is violated in cases where BC can. The somewhat surprising consequence of this is that neither algorithm "dominates" the other when used solely to halt search.

3.1 BC Doesn't Dominate E_t

Muscettola [1] describes an RTN for which BC does not prove the NTC is satisfied. This example is modified and reproduced in Figure 2. Initially the resource has 2 units available. BC will calculate a lower bound on $L_{min}^>(A)$ as follows: $U(A) = \{W, X, Y, Z\}$ and $A(A) = \{B, C, D\}$. Then $L_{min}^>(A) = -1$; it schedules W and Y before A , and schedules X and Z after A . Clearly, this schedule is not consistent with the original STN, so the bound calculated by BC is "loose" and BC incorrectly believes that more decisions are needed. The E_t algorithm is able to prove that the NTC is satisfied in this case by determining $L_{min}(t) \geq 0$ over the scheduling horizon. As a case in point, consider $L_{min}(10)$. The pending set $P(10) = \{B, C, D, X, Y, Z\}$. We see that it is possible to postpone B and X and achieve $L_{min}(10) = 0$. It is also possible to schedule C and Y at 10; however, scheduling C , for example, forces the scheduling of the producer B prior to C because of temporal constraints. This accounts for one of the consumptions, and makes it impossible to improve upon $L_{min}(10) = 0$. If we consider the flow problem in Figure 1 (b), we see that this is exactly what is calculated. The maximum flow for this network saturates the arcs to D, Z, C and Y . The only two events reachable in the residual graph are B and X ; thus $P_{max}^<(10) = \{B, X\}$. In addition, $L_{max}(t) \leq 2$ over the horizon, showing that the NTC is provably satisfied.

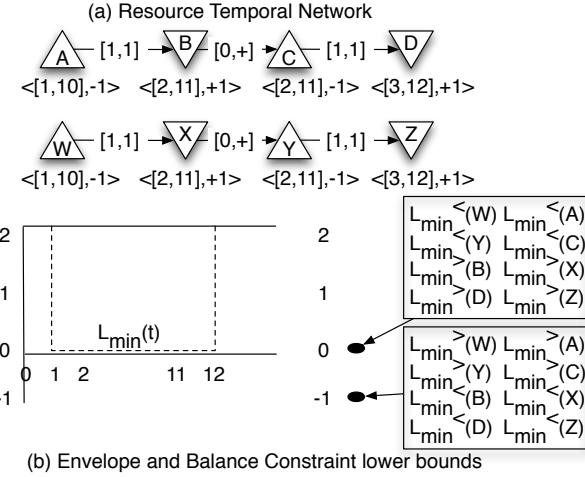


Fig. 2. An RTN for which BC fails to detect that the NTC is provably satisfied.

3.2 E_t Doesn't Dominate BC

Figure 3 describes an RTN for which E_t cannot show that the NTC is provably unsatisfied. Again, initially the resource has 2 units available. In this case, the BC will find $L_{max}^<(A) = L_{max}^<(B) = L_{max}^<(C) = 1$, but $L_{max}^<(D) = -5$, since $B(D) = \{A, B, C\}$. This proves that, no matter what additional constraints are imposed, the resource bound will be violated. By contrast, E_t cannot conclude that the NTC is provably unsatisfied. We see that B and C can be postponed until time 10, and A can be scheduled as early as 1, leading to $L_{max}(1) = 3$. At time 2, we could schedule D , but that would require scheduling everything before 2 with a resource availability of 0 being the result; at time 2 we can still find a schedule in which A and nothing else so $L_{max}(2) = 3$. At time 10, however, both consumption events must have taken place; in order to achieve the maximum resource available we can schedule D , leading to $L_{max}(11) = 0$. The calculation of $L_{min}(t)$ (not shown) provides no assistance in proving the NTC is violated. Not only does E_t fail to show that the NTC is provably violated, there are non-trivial ordering decisions that can be made; in the worst case, schedulers could spend a considerable amount of time fruitlessly continuing the search from states like the one shown in Figure 3.

4 A Tighter Balance Constraint

4.1 Setup

In order to achieve tighter bounds than either E_t or BC , we adopt a synthesis of both strategies. We use the same partition of events used by Laborie [6]. Like Laborie, we then find bounds on $L_{max}^<(X)$, $L_{min}^<(X)$, $L_{max}^>(X)$, and $L_{min}^>(X)$. Like Muscettola, we build a maximum flow problem whose residual graph is used to construct the supporting sets $P_{max}^<(X)$, $P_{min}^<(X)$, $P_{max}^>(X)$, and $P_{min}^>(X)$. The rules for constructing the flow problem are identical to those described previously; only the set of events defining the flow problems are different. To find $\Delta P_{max}^<(X)$ and $\Delta P_{min}^<(X)$, we solve a flow problem over the set of events $BS(X) \cup U(X)$. In these cases, $P_{max}^<(X) = \Delta P_{max}^<(X) \cup B(X)$ and $P_{min}^<(X) = \Delta P_{min}^<(X) \cup B(X)$. We define $\Delta P_{max}^{<C}(X) =$

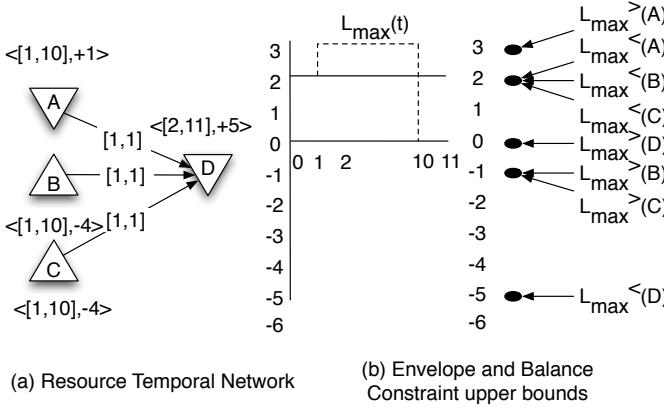


Fig. 3. An RTN for which E_t fails to detect that the NTC is provably unsatisfied.

$(BS(X) \cup U(X)) - \Delta P_{max}^<(X)$ and $\Delta P_{min}^{<C}(X) = (BS(X) \cup U(X)) - \Delta P_{min}^<(X)$. To find $\Delta P_{max}^>(X)$ and $\Delta P_{min}^>(X)$, we solve a flow problem over the set of events $AS(X) \cup U(X)$. In these cases, $P_{max}^>(X) = \Delta P_{max}^>(X) \cup S(X) \cup B(X) \cup BS(X)$ and $P_{min}^>(X) = \Delta P_{min}^>(X) \cup S(X) \cup B(X) \cup BS(X)$. We define $\Delta P_{max}^{>C}(X) = (AS(X) \cup U(X)) - \Delta P_{max}^>(X)$ and $\Delta P_{min}^{>C}(X) = (AS(X) \cup U(X)) - \Delta P_{min}^>(X)$. These supporting sets define schedules that prove the bounds are tight. For example, the resulting set $P_{max}^<(X)$ defines an STN constructed by adding the following constraints: $\forall V \in P_{max}^<(X) \{V < X\}$ and $\forall V \in P_{max}^{<C}(X) \{X \leq V\}$. We refer to the resulting bounds as the *Flow Balance Constraint (FBC)*, since it combines the features of the envelope with the flow-based approach to guarantee tightness. The number of flow problems to solve is $\leq 4n$, since we observe that if $X \in S(Y)$ then the same bounds apply for both X and Y .

FBC(\mathcal{R})

Enforce Arc Consistency on \mathcal{R}

for each event $X \in \mathcal{N}$

 Set up flow problems

 Bound(X)

end for

end

Bound(X)

 Find $\Delta P_{max}^<(X) \subset BS(X) \cup U(X)$

$$L_{max}^<(X) = \sum_{V \in \Delta P_{max}^<(X) \cup B(X)} c(V)$$

 Find $\Delta P_{min}^<(X) \subset BS(X) \cup U(X)$

$$L_{min}^<(X) = \sum_{V \in \Delta P_{min}^<(X) \cup B(X)} c(V)$$

 Find $\Delta P_{max}^>(X) \subset AS(X) \cup U(X)$

$$L_{max}^>(X) = \sum_{V \in \Delta P_{max}^>(X) \cup B(X) \cup BS(X) \cup S(X)} c(V)$$

 Find $\Delta P_{min}^>(X) \subset AS(X) \cup U(X)$

$$L_{min}^>(X) = \sum_{V \in \Delta P_{min}^>(X) \cup B(X) \cup BS(X) \cup S(X)} c(V)$$

end

Fig. 4. A sketch of the Flow Balance Constraint.

The algorithm for *FBC* is described in Figure 4. As is done in [6], we maintain the partition of the STN relative to each event X during the arc-consistency enforcement phase. We now proceed to prove that the resulting bounds on quantities like $L_{max}^<(X)$ are tight. To do so, we first show that there is at least one schedule justifying the bound, and then show that there is no better bound than that found using the flow problem.

Theorem 1. *Let R be an RTN and X be an event in R . Suppose $\Delta P_{max}^<(X) \neq \emptyset$. Let R' be the STN formed by adding the following constraints to R : $\{V \leq X\}$ for all $X \in \Delta P_{max}^<(X)$ and $\{X > V\}$ for all $X \in \Delta P_{max}^{<C}(X)$. Then R' has at least one temporally consistent solution.*

Proof. Since $V \in P(X)$ the imposition of a single constraint alone doesn't make the STN inconsistent. Imposing a constraint $V \leq X$ only decreases V_{ub} and increases X_{lb} . Since $\Delta P_{max}^<(X)$ is a predecessor set, all $\{V \leq X\}$ can be imposed simultaneously without impacting consistency. Imposing a constraint $X < V$ only decreases X_{ub} and increases V_{lb} . Since $\Delta P_{max}^{<C}(X)$ is a successor set, all $\{X > V\}$ can also be imposed simultaneously without impacting consistency. Finally, X is the only event whose bounds are acted on by both classes of constraint. Consider two events A, B . Since $A \in \Delta P_{max}^<(X)$ and $B \in \Delta P_{max}^{<C}(X)$ we know it can't be the case that $B < A$. But then either $A \leq B$ or A and B can be ordered in any way, and we already know X can be ordered any way with respect to A or B . Thus, $A \leq X < B$ is possible and no such ordering prevents other linearizations with respect to X . \square

Theorem 2. *Let R be an RTN and X be an event in R . Suppose $\Delta P_{max}^<(X) \neq \emptyset$. Then $\sum_{V \in \Delta P_{max}^<(X) \cup B(X)} c(V)$ is the maximum possible value of $L_{max}^<(X)$.*

Proof. Since we construct the flow problems in exactly the same way as is done in [1], we state this as a corollary of Theorem 1 of [1]. \square

The proofs for the tightness of the bounds on $L_{max}^<(X)$, $L_{min}^<(X)$ and $L_{min}^>(X)$, are similar.

4.2 Relating E_t and FBC

In this section, we formally establish the relationship between the intervals over which the *FBC* and E_t bounds hold. Intuitively, $L_{max}^<(X)$ is the maximum availability of the resource for an interval of time immediately prior to the time X happens. What is the interval of time over which this value holds? Suppose we find a set $P_{max}^<(X)$ that supports $L_{max}^<(X)$. Then as previously stated, this bound assumes that we impose the following constraints: $\forall V \in P_{max}^<(X) \{V < X\}$ and $\forall V \in P_{max}^{<C}(X) \{X \leq V\}$ to get a new RTN R' . Note that the new unordered set $U'(X) = \emptyset$, and $BS'(X) = \emptyset$, and the new closed set $B'(X) = P_{max}^<(X)$. Note also that $L_{max}^<(X)$ is the resource availability no earlier than $V_{ub'}^* = \max_{V \in B'(X)} V_{ub'}$. Due to the new constraints X may have a new upper bound $X_{ub'}$, which defines the latest time that $L_{max}^<(X)$ holds. Finally, note that $L_{max}^<(X) = L_{max'}(t)$ over the interval $[V_{ub'}^*, X_{ub'} - 1]$. An example is shown in Figure 5. We now can demonstrate the precise relationship between $L_{max}(t)$ and $L_{max}^<(X)$.

Theorem 3. $L_{max}^<(X) \leq L_{max}(t)$ for $V_{ub'}^* \leq t \leq X_{ub'} - 1$.

Proof. The time bounds are explained in the previous paragraph. Since we add new constraints, $\forall t L_{max'}(t) \leq L_{max}(t)$. Since $L_{max}^<(X) = L_{max'}(t)$ we're done. \square

Now suppose we find a set $P_{max}^>(X)$ that supports $L_{max}^>(X)$. Then as previously stated, this bound assumes that we impose the following constraints: $\forall V \in P_{max}^>(X) \{V \leq X\}$ and $\forall V \in P_{max}^{>C}(X) \{X < V\}$ to get a new RTN R' . Note that the new pending set $U'(X) = \emptyset$, and $B'(X) \cup S'(X) \cup BS'(X) = P_{max}^>(X)$. Note also that $L_{max}^>(X)$ is the resource availability no later than $W_{lb'}^* = \min_{W \in R - (B'(X) \cup BS'(X) \cup S'(X))} W_{lb'}$. Due to the new constraints X may have a new lower bound $X_{lb'}$, which defines the earliest time that $L_{max}^>(X)$ holds. Finally, note that $L_{max}^>(X) = L_{max'}(t)$ over the interval $[X_{lb'}, W_{lb'}^* - 1]$.

Theorem 4. $L_{max}^>(X) \leq L_{max}(t)$ for $X_{lb'} \leq t \leq W_{lb'}^* - 1$.

Proof. Identical to the previous proof. \square

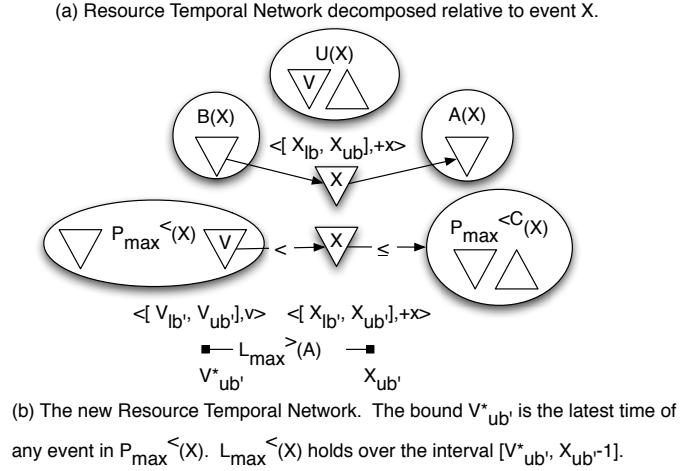


Fig. 5. Deriving the time intervals over which resource availability bounds hold. The example shows the calculation for $L_{max}^<(X)$.

We show an example of the relationship in Figure 6. The RTN is the same as that in Figure 3. This example also shows why BC can answer positively in cases where E_t cannot. In this case, BC finds a tight bound on $L_{max}^<(D)$ that proves that the NTC cannot be satisfied. Unfortunately, E_t must calculate the maximum over all of the event-based bounds that hold at a time t , and cannot do better than return $?$ for this RTN.

5 Dominance

In this section we will describe our dominance criteria and show that FBC dominates E_t and BC ; we do not formally show that BC and E_t do not dominate each other, relying on the intuition of Section 3 to adequately demonstrate this. As previously stated, we assume that FBC , E_t and BC are used to detect whether a scheduling algorithm can halt or must continue. In order to motivate our definition of dominance, suppose that some resource bounding procedure A is used in a chronological search framework with a static variable and value ordering heuristic. Then we would like A to dominate B if A leads to smaller search trees than using B . Now suppose A and B are used in a stochastic sampling approach where we sample RTNs that are temporally consistent with the original problem. Then we would like A to dominate B if A leads to less sampling than B . We use the following definition of dominance.

Definition 1. Let R be an RTN. Let $T(R)$ be the set of all temporally consistent RTNs that can be formed from R by adding temporal constraints to R . Let $A, B : R \rightarrow \{T, F, ?\}$. Let $U_A(R) = S \in T(R) | A(S) = ?$. Then A dominates B on R if $U_A(R) \subset U_B(R)$. A dominates B if $\exists R$ such that A dominates B on R and there exists no S such that B dominates A on S . We write $A \prec_R B$ or $A \prec B$ as appropriate.

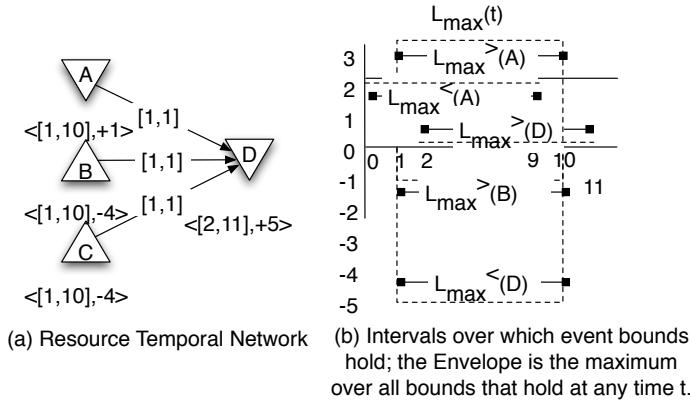


Fig. 6. An RTN for which E_t fails to detect that the NTC is unsatisfiable.

Theorem 5. $FBC \prec BC$.

Proof. Theorems 1 and 2 show that the flow construction guarantees the tightest possible bounds on $L_{\max}^<(X)$, $L_{\max}^>(X)$, $L_{\min}^<(X)$ and $L_{\min}^>(X)$ for any RTN. Thus there can be no S such that $BC \prec_S FBC$. The example shown in Figure 2 shows at least one RTN R for which $FBC \prec_R BC$. This completes the proof. \square

Define $E^<(t) = X | L_{\max}^<(X)$ holds at t and $E^>(t) = X | L_{\max}^>(X)$ holds at t .

Theorem 6. $L_{\max}(t) = \max(\max_{X \in E^<(t)} L_{\max}^<(X), \max_{X \in E^>(t)} L_{\max}^>(X))$ and $L_{\min}(t) = \min(\min_{X \in E^<(t)} L_{\min}^<(X), \min_{X \in E^>(t)} L_{\min}^>(X))$.

Proof. Corollary of Theorems 3 and 4 \square

Theorem 7. $FBC \prec E_t$

Proof. Theorem 6 shows that there is no RTN S for which $E_t \prec_S FBC$. The example in Figure 3 shows that there is at least one RTN R for which $FBC \prec_R E_t$. This completes the proof. \square

6 Complexity of Calculating FBC

A naive algorithm for calculating FBC builds a new flow network for each event $X \in \mathcal{N}$. An equally naive analysis of the complexity of this algorithm is $O(n * \text{MaxFlow}(n, m))$, where m is the number of anti-precedence relationships in the RTN. However, this is unsatisfactory for a number of reasons. Consider the RTN in Figure 2. The flow networks required to calculate FBC are *identical* for A, B, C, D because $S(A) = S(B) = S(C) = S(D)$. Additionally, the flow networks include only a small fraction of the n nodes in the RTN; strict precedences and equalities among events will generally reduce the size of the flow problems. These observations suggest it might be cheaper than $O(n * \text{MaxFlow}(n, m))$ to calculate FBC .

6.1 A Nontrivial Lower Bound

In this section we provide a *lower* bound on the complexity of the naive approach to calculating FBC . We do so by constructing an RTN such that the flow problem to solve for each event is both non-trivial and distinct. The RTN is a "square" graph with \sqrt{n}

events per side. We index events by row and column in the square. The RTN has the following strict precedences: $(i, j) < (i + 1, j)$, and $(i, j) < (i, j + 1)$ (obviously omitting those links for which the indices are outside the bounds $[0, \sqrt{n}]$). By construction, $P((i, j)) = ((x, y) | x < i \wedge y > j) \cup ((u, v) | u < i \wedge v > j)$. Thus, all of the flow graphs are distinct. Every such set has a non-trivial part of the flow graph. Notice that we can assign $c(X)$ arbitrarily to the events of the RTN, as long as they are all non-zero and there is a mix of consumers and producers. This RTN is shown in Figure 7.

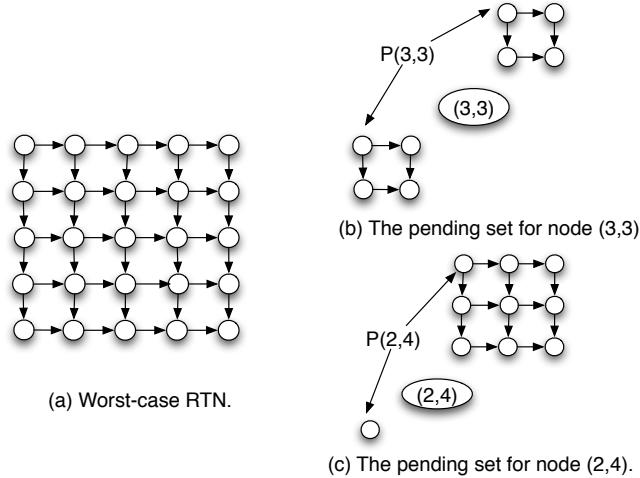


Fig. 7. A perverse RTN providing a worst-case lower bound for calculating FBC .

We now proceed to construct a lower bound on the complexity of the naive approach for calculating FBC on this RTN. By construction we have guaranteed that no "quick fixes" can be used to decrease the complexity. The larger of the two induced flow problem for event (i, j) contains $\max((i - 1)(\sqrt{n} - j - 1), (j - 1)(\sqrt{n} - i - 1))$ events, and at least this many flow arcs (we could do an exact count but it isn't necessary since we're providing a lower bound.) Let us now assume we are using a FIFO preflow-push algorithm to solve each flow problem [10]; this ignores any efficiency gained from analyzing the pushable flow, but is also suitable for our purposes. If v is the number of nodes in the flow problem, there are at least v edges, and so the complexity is $\Omega(v^{2.5})$. Using these assumptions the total complexity of solving all the flow problems is

$$\sum_{j=1}^{\sqrt{n}} \sum_{i=1}^{\sqrt{n}} \max((i - 1)(\sqrt{n} - j - 1), (j - 1)(\sqrt{n} - i - 1))^{2.5}$$

First, we simplify the sum to get a lower bound:

$$\leq \sum_{j=0}^{\sqrt{n}-1} \sum_{i=0}^{\sqrt{n}-1} (i(\sqrt{n} - j))^{2.5}$$

We next approximate the sum with the integral (which, while bounding above, is close enough for our purposes):

$$\approx \int_{j=0}^{\sqrt{n}-1} \left(\int_{i=0}^{\sqrt{n}-1} (i(\sqrt{n} - j))^{2.5} di \right) dj$$

The first integral with respect to i is trivial. Substituting for $\sqrt{n} - j$, we see that

$$\int (\sqrt{n} - j)^{2.5} dj = (\sqrt{n} - j)^{2.5} \left(\frac{j - \sqrt{n}}{3.5} \right)$$

Ultimately we have

$$= \left(\frac{(\sqrt{n} - 1)^{3.5}}{3.5} \right) \left(\frac{\sqrt{n}^{3.5}}{3.5} - \frac{1}{3.5} \right)$$

Collecting the high order positive powers of \sqrt{n} we see that the naive algorithm has a lower bound $\Omega(n^{3.5})$. Thus, for preflow-push flow algorithms using FIFO queues, the naive algorithm for FBC is $\Omega(n * \text{MaxFlow}(n, m))$.

6.2 Incrementally Calculating FBC

As stated previously, the incremental technique described in [7] shaves a factor of n off the cost of calculating E_t . To do so, the incremental approach in [7] relies on restricted modifications of the flow problem. If A is in the flow problem, it may be removed if it has no successors other than the sink remaining in the flow problem; if B is not in the flow problem, it may be added if it has no predecessors other than the source in the flow problem. This provides some hope that we can find a way to eliminate the factor of n "extra" cost for calculating FBC that we described in the previous section. Unfortunately, this is not the case. The crucial element of the complexity analysis in [7] requires that an event always move from open to pending to closed. We show that naively applying the incremental algorithm may result in a non-trivial number of events moving from closed to pending, thereby defeating the cost-savings measures. Consider the RTN described in Figure 7. The longest chain of events that could benefit from incremental solving of flow problems is $O(\sqrt{n})$, and there are $O(\sqrt{n})$ chains of events we must solve for which the incremental approach cannot save any computation. Each of these induces a total complexity of $O(\text{MaxFlow}(n, m))$. Thus, for this problem, even the incremental approach to solving flow problems customized for the E_t algorithm cannot reduce the complexity of FBC below $\Omega(\sqrt{n} \text{MaxFlow}(n, m))$.

We can take advantage of the incremental flow calculation by judicious ordering of the bounds we calculate for an event X . For example, to find $\Delta P_{\max}^<(X)$ we solve a flow problem over $BS(X) \cup U(X)$. To find $\Delta P_{\max}^>(X)$ we solve a flow problem over $AS(X) \cup U(X)$. The change in the flow problem to be solved allows the incremental approach; thus, we can use the solution of the flow that gives us $\Delta P_{\max}^<(X)$ in order to find $\Delta P_{\max}^>(X)$. The total complexity of solving both flow problems is $O(\text{MaxFlow}(a, b))$ where $a = |BS(X) \cup U(X) \cup AS(X)|$ which is cheaper than solving both flows separately as long as $U(X) \neq \emptyset$.

We can also take advantage of the incremental flow calculation by judicious ordering of the calculation of event bounds. To see how this works, we first observe that we only need to consider the evolution of the flow problems for $\Delta P_{\max}^<(X)$ and $\Delta P_{\min}^<(X)$ (since we handled the incrementality of $\Delta P_{\max}^<(X)$ and $\Delta P_{\max}^>(X)$ in the previous paragraph). Suppose we are calculating the bounds for X and Z 's bounds were previously calculated. Suppose $Z \in B(X) \cup BS(X) \cup S(X)$. Then $A \in BS(Z)$ must be either in $B(X)$ or $BS(X)$ and $B \in U(Z)$ must be either in $B(X)$, $BS(X)$ or $U(X)$. This fits the conditions required to use the incremental approach by reusing the solution to the flow problem on $U(Z) \cup BS(Z)$ and have a chance at successfully reducing the time. If we had *cached* this flow problem, we could then retrieve it in linear time and reuse it. The total complexity of solving both flow problems is $O(\text{MaxFlow}(a, b))$ where $a = |BS(X) \cup U(X) \cup BS(Z) \cup U(Z)|$. Note if there are no shared events in the two flow problems, there is no savings; but if there are shared events, then the second flow problem could be solved faster because of the reuse of the previous flow solution.

For some event X there might be numerous previously calculated bounds that could be used. The simplest policy is to use the bounds generated by the event Z that occurs

closest to X and satisfies the conditions described above. We call the resulting algorithm $FBC - DFS$, and it is described in Figure 8 below. We assume that X_i is a topological sort of the events; as before, we need to calculate bounds for only one event in an equivalence class.

```

FBC-DFS( $\mathcal{R}, X_i$ )
  Enforce Arc-Consistency on  $\mathcal{R}$ 
   $F_{max} = F_{min} = \emptyset$ 
  for each event  $X_i$ 
    Find the latest  $Y$  previously visited such that  $Y \in B(X_i) \cup BS(X_i) \cup S(X_i)$ 
    if  $Y \neq X_i$  Pop  $F_{max}$  and  $F_{min}$  to  $Y$ 
    Build flow problems for  $L_{max}^<(X)$  from  $top(F_{max})$  and  $L_{min}^<(X)$  from  $top(F_{min})$ 
    Inc-Bound( $X_i$ )
    Push the flow solutions for  $L_{max}^<(X)$  onto  $F_{max}$  and  $L_{min}^<(X)$  onto  $F_{min}$ 
  end for
end

Inc-Bound( $X$ )
  Find  $\Delta P_{max}^<(X) \subset BS(X) \cup U(X)$ 
   $L_{max}^<(X) = \sum_{V \in \Delta P_{max}^<(X) \cup B(X)} c(V)$ 
  Find  $\Delta P_{min}^<(X) \subset BS(X) \cup U(X)$ 
   $L_{min}^<(X) = \sum_{V \in \Delta P_{min}^<(X) \cup B(X)} c(V)$ 
  Build flow problems for  $L_{max}^>(X), L_{min}^>(X)$ 
  Find  $\Delta P_{max}^>(X) \subset AS(X) \cup U(X)$ 
   $L_{max}^>(X) = \sum_{V \in \Delta P_{max}^>(X) \cup B(X) \cup BS(X) \cup S(X)} c(V)$ 
  Find  $\Delta P_{min}^>(X) \subset AS(X) \cup U(X)$ 
   $L_{min}^>(X) = \sum_{V \in \Delta P_{min}^>(X) \cup B(X) \cup BS(X) \cup S(X)} c(V)$ 
end

```

Fig. 8. A sketch of the Flow Balance Constraint implemented by using a Depth-First search through the precedence graph and using an incremental approach to reduce computation time for constructing the bounds.

Theorem 8. Let \mathcal{R} be an RTN with m induced anti-precedence constraints in its STN S . Let W be the partial order width and L be the partial order length of the partially ordered set induced by the anti-precedence graph. Then Algorithm $FBC - DFS$ takes $O(W * \text{MaxFlow}(n, m))$ time and $O(Lm)$ space.

Proof. There is a cover of the partial ordered set consisting of W chains. Assume that the order of processing the event bounds is consistent with any permutation of these chains. Each chain is a sequence such that $X_{i-1} \in B(X_i) \cup BS(X_i) \cup S(X_i)$. Then, for each chain, we can reuse the flow solutions from $top(F_{max})$ and $top(F_{min})$ to solve each X_i ; the total incremental cost for each chain is $O(\text{MaxFlow}(n, m))$. Since there can't be more than W such chains, we're done with this part of the proof. Since the longest such chain is of length L , we store at most L copies of a flow problem on at most n nodes and m edges.

7 Higher Order Balance Constraint

The quantities $L_{max}^<(X), L_{max}^>(X), L_{min}^<(X)$ and $L_{min}^>(X)$ can be thought of as *first order* checks on resource availability, in that they calculate resource bounds before and after one event. In this section we generalize these techniques in order to calculate higher-order resource availability checks after k events. To see why this is valuable, consider Figure 9. We see that no first order bound calculated by FBC proves that this RTN is impossible to solve. However, notice that we can show that the *maximum* available resource after both A and B have occurred is -1 . This corresponds to one

of two schedules: it is possible to either schedule $D \leq B$ or $C \leq A$, but not both simultaneously. Thus, without further search, we can prove that the NTC is unsatisfied.

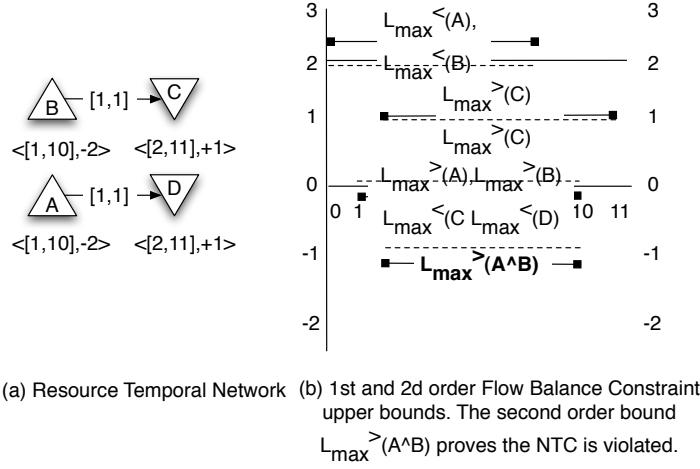


Fig. 9. An RTN for which FBC fails to prove that the NTC is unsatisfied.

This example shows that it may be valuable to perform 2^d -order checks on resource availability by determining resource availability immediately before and after sets of 2 events. In order to do this for $L_{max}^<(X \wedge Y)$ we must account for the following possibilities: neither X nor Y have occurred, X has occurred but Y has not, and vice versa. First, we solve the flow problem over the events of $(BS(X) \cup U(X)) \cap (BS(Y) \cup U(Y))$ to find the maximum availability strictly before both X and Y . We call this set of events $\Delta P_{max}^<(\neg(X \wedge Y))$. The second requires adding the constraint $X < Y$, finding the new set of closed events $B(X < Y)$, and then solving the flow problem defined by $BS(X < Y) \cup U(X < Y)$ and find the maximum availability strictly before Y . Call this set $\Delta P_{max}^<(X < Y)$. The last requires adding the constraint $Y < X$, finding the new set of closed events $B(Y < X)$, and then solving the flow problem defined by $BS(Y < X) \cup U(Y < X)$ and find the maximum availability strictly before X . Call this set of events $\Delta P_{max}^<(Y < X)$. We define $L_{max}^<(X \wedge Y)$ as

$$\begin{aligned} \max(& \sum_{V \in (C(X) \cap C(Y)) \cup \Delta P_{max}^<(\neg(X \wedge Y)) \cup B(X) \cap B(Y)} c(V), \\ & \sum_{V \in C(X < Y) \cup \Delta P_{max}^<(X < Y) \cup B(X < Y)} c(V), \\ & \sum_{V \in C(Y < X) \cup \Delta P_{max}^<(Y < X) \cup B(Y < X)} c(V)) \end{aligned} \quad (1)$$

For $L_{max}^>(X \wedge Y)$ we first solve the flow problem over the events of $(AS(X) \cup U(X)) \cap (AS(X) \cup U(X))$. Call the resulting set $\Delta P_{max}^>((X \wedge Y))$. Now we must also include events in $B(X) \cup BS(X) \cup S(X) \cup B(Y) \cup BS(Y) \cup S(Y)$. We thus define $L_{max}^>((X \vee Y)) = \sum_{V \in \Delta P_{max}^>((X \wedge Y)) \cup B(X) \cup BS(X) \cup S(X) \cup B(Y) \cup BS(Y) \cup S(Y)} c(V)$. This

is the amount after the schedule assuming both X and Y have occurred. The lower bounds are calculated in a similar manner.

To see how this works in Figure 9, note $U(A) \cap U(B) = \emptyset$. The maximum available under these circumstances is achieved by postponing all the events, which is 2. If we impose $A < B$, $P(B) = D$; the best schedule here is A, D for an availability of 1. The same applies if $B < A$. Thus, $L_{max}^<(X \wedge Y) = 1$. To calculate $L_{max}^>(A \wedge B)$ we observe that the first incremental calculation leads to $2 - 4 = -2$ since neither A nor B are assumed to have occurred. If we impose $A < B$, the max involves A ; but we must assume B has happened as well to achieve the maximum, which leads to $1 - 2 = -1$. The same applies for impose $B < A$. The result leads to the schedule defined by ensuring all of the events are before or equal to A and B . We can define the interval over which the bounds hold as we have previously.

The total complexity of the resulting naive algorithm for calculating *Second order Flow Balance Constraint* (FBC^2) is $\Omega((n^2)MaxFlow(n, m))$. The n^2 term comes from the fact that $n(n - 1)$ pairs of bounds must be calculated; the complexity bounds obscure the fact that 4 flow problems must be solved per pair of events.

Note, however, that n^2 is a very crude estimate of the total number of bounds to compute. If A strictly precedes B then $P_{max}^<(A \wedge B) = P_{max}^<(B)$. Thus, the induced precedences vastly reduce the number of bounds to calculate. Additionally, the sizes of the flow problems will generally be larger as the number of events involved climbs. These factors make a more precise complexity analysis difficult. Finally, it is likely that the incremental flow algorithm described in [7] can be used to further reduce the complexity. It is sufficient for our purposes to demonstrate that even tighter inferred constraints can be calculated in time polynomial in the number of events considered.

We can further generalize this to sets of k events in the same manner. The complexity of the naive algorithm for calculating FBC^k is $\Omega\left(\binom{n}{k}(2^k)MaxFlow(n, m)\right)$. This is because there are $\binom{n}{k}$ bounds to calculate, and each bound requires solving 2^k flow problems (as well as arc-consistency enforcement steps).

8 Conclusions and Future Work

In this paper we have shown, contrary to expectations, that BC and E_t are not strictly comparable in terms of their power to halt search over partial ordered schedules for RTNs. We have also shown how to exploit the features of BC and E_t to construct FBC , a tighter bound on the availability of resources for RTNs than either of the previous approaches. The resulting bound can be computed by the algorithm $FBC - DFS$ in $O(MaxFlow(n, m))$ time, but $n * O(MaxFlow(n, m))$ space. When used in identical search algorithms with identical static variable and value orders, FBC will generate search trees with less than or equal nodes than either BC or E_t . The technique generalize for calculating FBC leads to even tighter bounds, but at sharply increased computational cost.

While we have proven dominance of FBC , an empirical study will be necessary to determine whether it is worth the overhead. An empirical study will also have the added benefit of shedding more light on the relative value of E_t and BC for speeding up the solution of scheduling problems. There are numerous possibilities for speeding up the solution of the flow problems necessary to calculate the bounds. The DFS-FBC algorithm could benefit from judicious node orderings, either to reduce total flow problem solving costs, or to reduce storage costs. A second, equally important empirical study will be necessary to shed light on how to integrate heuristics that make use of the various bounds. Laborie [6] has built numerous such heuristics for BC , providing a good starting point. However, such heuristics likely have complex interactions with the pruning power of the envelopes. It will likely be necessary to trade off between the pruning

power and heuristic predictiveness of the resource bounds to craft the best scheduling algorithm.

Changes to dominance criteria that we use are worth contemplating. On the one hand, requiring that A dominates B on R if $U_A(R) \subset U_B(R)$ is rather strong, and could be weakened, say, to $|U_A(R)| < |U_B(R)|$. On the other hand, requiring that A dominates B if $\exists R$ such that A dominates B on R and there exists no S such that B dominates A on S is somewhat weak, and perhaps could be strengthened.

For simplicity, we concentrated here on analyzing the complexity of *FBC* using Preflow-Push with FIFO queues. Generalizing the complexity analysis to other flow algorithms may be worthwhile, but complex. In addition, it may be possible to improve upon the complexity analysis of *FBC*. Recall from Dilworth's theorem that $n \leq LW$; aesthetically appealing though this is, it is not very helpful, since W can grow faster than \sqrt{n} for some classes of problems. However, there may be useful relationships between L, W, n and m for some classes of problems that can lead to tighter complexity bounds.

References

1. Muscettola, N.: Computing the envelope for stepwise-constant resource allocations. In: Proceedings of the 8th International Conference on the Principles and Practices of Constraint Programming. (2002) 139 –154
2. Laborie, P.: Resource temporal networks: Definition and complexity. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. (2003) 948 – 953
3. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. *Artificial Intelligence* **49** (1991) 61–94
4. Jónsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in interplanetary space: Theory and practice. In: Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling. (2000)
5. Morris, P., Muscettola, N., Tsamardinos, I.: Reformulating temporal plans for efficient execution. In: Proceedings of the 15th National Conference on Artificial Intelligence. (1998)
6. Laborie, P.: Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence* **143** (2003) 151–188
7. Muscettola, N.: Incremental maximum flows for fast envelope computation. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling. (2004)
8. Chapman, D.: Planning for conjunctive goals. *Artificial Intelligence* **32** (1987) 333 – 377
9. Kumar, T.K.S.: Incremental computation of resource-envelopes in producer-consumer models. In: Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming. (2003) 664 – 678
10. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall (1993)