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 compare existing techniques to bound the resource availability of partially ordered events. We first show that, contrary to intuition, two existing techniques, one due to Laborie and one due to Muscettola, are not strictly comparable in terms of the size of the search trees generated under chronological search with a fixed heuristic. We describe a generalization of these techniques called the *Flow Balance Constraint* to tightly bound the amount of available resource for a set of partially ordered events with piecewise constant resource impact. We prove that the new technique generates smaller proof trees under chronological search with a fixed heuristic, at little increase in computational expense. We then show how to construct tighter resource bounds but at increased computational cost.

1 Introduction

Scheduling is about simultaneously satisfying temporal constraints and resource availability constraints. Following Muscettola [1], Laborie [2] has provided a simple but expressive formalism for such 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 kn:AIPS) 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.

A simple resource use model assumes all resources impacts occur instantly; this is known as the piecewise constant resource impact model. Under this assumption, 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. While resource bounds are also important inputs to heuristics to drive search 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 a generalization of these techniques to tightly bound the amount of available resource for a set of partially ordered events called the *Flow Balance Constraint* (FBC). FBC is a synthesis of the approaches described in [6, 1]. We prove that FBC generates smaller proof trees under chronological search. In Section 7 we exploit the results of [7] to calculate FBC incrementally, thereby reducing its computational cost. In Section 8 we then generalize FBC in order to construct even tighter resource bounds but at increased computational cost. Finally, in Section 9 we conclude and describe future work.

2 Previous Work

In this paper we will assume that time is represented using integers[1]. 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 lower-than constraint and one greater-than constraint on the resource, and that the greater-than constraint imposes a minimum resource availability of 0. 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 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 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 below 0, and the minimum calculated availability of the resource is always a lower bound on the actual minimum available resource and is above the resource upper bound. Otherwise, the satisfiability state 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 $S(\mathcal{X})$ 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 $\mathcal{T}(\mathcal{X})$ such that if $X \in S$ then every event Y such that $Y > X$ is also in S. Let R be an RTN and let $A(R)$ be any procedure for evaluating

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

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.

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] takes a different approach to bounding resource availability. The approach is to find 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 are *Closed* $C(t)$, *Pending* $P(t)$, or *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 X anti-precedes Y 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 $\Delta P^{C}_{max}(t) = P(t) - \Delta P_{max}(t)$. 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 $\{Y_{ub} > t\} \forall Y \in \Delta P^{C}_{max}(t) \cup O(t)$ 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 $I = \{\forall X X_{lb} \cup X_{ub}\}$ 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. We will usually refer to the unique instants. We will also refer to $e(i) = \{i|X_{lb} = i \cup Y|Y_{ub} = i\}$, those events that define i in I. The complexity of the algorithm described in [1] is $O(n * MaxFlow(n, m))$, where $n = |\mathcal{N}|$ and m is the number of anti-precedence relationships in the arc-consistent STN. In [7] this is reduced to $O(MaxFlow(n, m))$ by taking advantage of the order in which edges are added to and removed from a single maximum flow problem. The

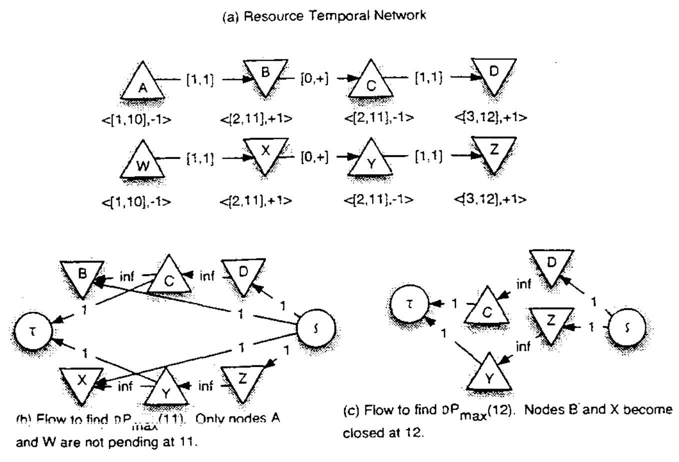

(a) Resource Temporal Network
A
[1,1]
B
[0,+]
C
[1,1]
D
<[1,10],-1>
<[2,11],+1>
<[2,11],-1>
<[3,12],+1>
W
[1,1]
X
[0,+]
Y
[1,1]
Z
<[1,10],-1>
<[2,11],+1>
<[2,11],-1>
<[3,12],+1>
inf
1
τ
σ
(b) Flow to find DP_max(11). Only nodes A and W are not pending at 11.
(c) Flow to find DP_max(12). Nodes B and X become closed at 12.

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

crucial observation is that when computing E_t an event always move from open to pending to closed and stays closed. As shown in Figure 1, this guarantees that arcs and vertices of the flow problem are removed from the sink τ and added to the source σ of 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 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 reason 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". Thus, 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 $L_{min}(11)$. The pending set $P(11) = \{B, C, D, X, Y, Z\}$. We see that it is possible to postpone B and X and achieve $L_{min}(11) = 0$. It is also possible to schedule C and Y, the other consumers; 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 $L_{min}(11) = 0$. If we consider the flow problem in Figure 1 (b), we see that this is exactly what is calculated. In addition, $L_{min}(t) \leq 2$ over the horizon, showing that the NTC is provably satisfied.

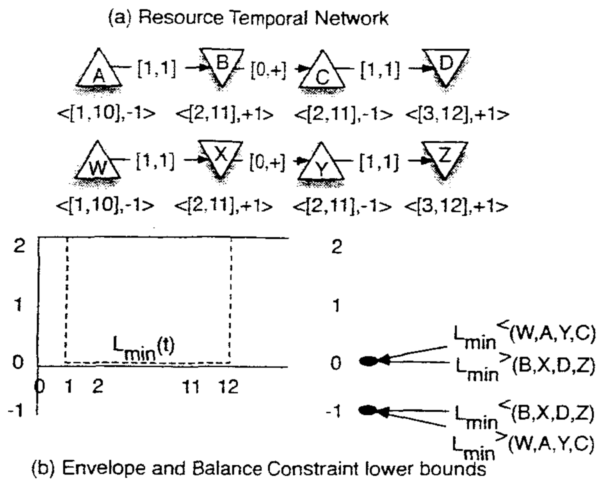


Fig. 2. An RTN for which BC fails to detect that the problem is solved.

3.2 E_t Doesn't Dominate BC

Figure 3 describes an RTN for which E_t cannot show that the NTC is provably not satisfied. 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 NTC will be violated. By contrast, E_t cannot conclude that the NTC is violated. We see that B and C can be postponed until time 10, and A can be scheduled early, 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 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 following partition of the STN relative to X: $C(X)$ is the set of events that must have been scheduled before X, $O(X)$ is the set of events that must be scheduled after X, $E(X)$ is the set of events that must occur at the same time as X, and $P(X)$ is the remainder of the events. Like Laborie, we then find $L^{<}_{max}(X)$, $L^{<}_{min}(X)$, $L^{>}_{max}(X)$, 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 considered in the flow problem is different. The resulting set $P^{<}_{max}(X)$ defines an STN constructed by adding

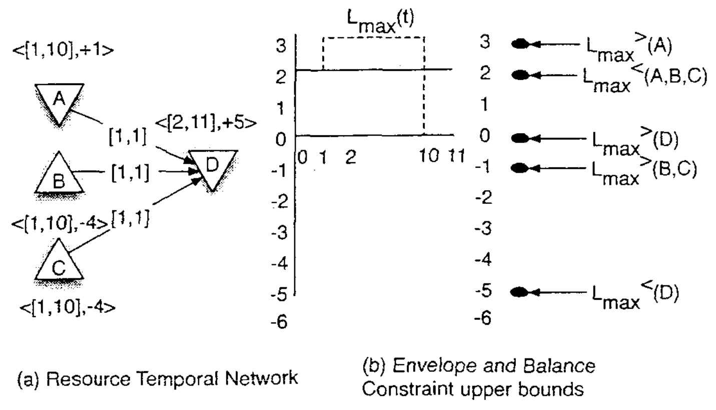


Fig. 3. An RTN for which E_t fails to detect that the problem is unsolvable.

the following constraints: $\forall V \in P^{<}_{max}(X)\{V < X\}$ and $\forall V \in P^{<C}_{max}(X)\{X \leq V\}$. Note that we also need find $\Delta P^{<}_{max}(X) \subset P(X)$ and so $P^{<}_{max}(X) \subset P(X) \cup C(X)$. The other set containment properties similar. 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.

One might think from this analysis that the number of flow problems to solve is $4n$, i.e. one flow problem for each of $L^{<}_{min}(X), L^{>}_{min}(X), L^{<}_{max}(X), L^{>}_{max}(X)$. Somewhat surprisingly, this is not necessary; only $2n$ flow calculations are needed. Consider $P(X)$. By definition, X has *no* anti-precedence arcs to any event in $P(X)$; in the language of [1] it is flow-isolated. X doesn't contribute to $L^{<}_{min}(X), L^{<}_{max}(X)$ and does contribute to $L^{>}_{min}(X), L^{>}_{max}(X)$; once the flow problems are solved, we simply add $E(X)$'s resource impact. The sets we find from solving the flows are thus referred to as $\Delta P^{e}_{max}(X)$ and $\Delta P^{e}_{min}(X)$

```
FBC($\mathcal{R}$)
   Make the STN $S$ arc-consistent.
   Infer all anti-precedence and precedence links. (transitive closure)
   Collect all sets $P(X), C(X)$.
   for each event $X$
      Build flow problems for upper and lower bounds from $P(X)$
      Bound($X$)
   end for
end

Bound($X$)
   Find $\Delta P^{e}_{max}(X) \subset P(X)$
   $L^{>}_{max}(X) = \sum_{V \in \Delta P^{e}_{max}(X) \cup C(X) \cup E(X)} c(V)$
   $L^{<}_{max}(X) = \sum_{V \in \Delta P^{e}_{max}(X) \cup C(X)} c(V)$
   Find $\Delta P^{e}_{min}(X) \subset P(X)$
   $L^{>}_{min}(X) = \sum_{V \in \Delta P^{e}_{min}(X) \cup C(X) \cup E(X)} c(V)$
   $L^{<}_{min}(X) = \sum_{V \in \Delta P^{e}_{min}(X) \cup C(X)} c(V)$
end
```

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

The algorithm for FBC is described in Figure 4. Since we must derive the transitive closures of the anti-precedence and strict precedence constraints, we can collect the sets $P(X), E(X), C(X)$ in time proportional to m the number of induced anti-precedences in the graph. Because we find $\Delta P^e_{max}(X)$ and $\Delta P^e_{min}(X)$ and using maximum flow, this is arguably the most expensive part of the algorithm; in the next section we discuss complexity in more detail.

We now proceed to prove that the resulting bounds on quantities like $L^<_{max}(E)$ 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 $\Delta P^e_{max}(X) \neq \emptyset$. Let R' be the STN formed by adding the following constraints to R: $\{V \leq X\}$ for all $X \in \Delta P^e_{max}(X)$ and $\{X > V\}$ for all $X \in \Delta P^C_{max}(X)$. 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 $\Delta P^e_{max}(X)$ 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 $\Delta P^C_{max}(X)$ 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 $A \in \Delta P^e_{max}(X)$ and $B \in \Delta P^C_{max}(X)$ 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. □

Theorem 2. *Let R be an RTN and X be an event in R. Suppose $\Delta P^e_{max}(X) \neq \emptyset$. Then $\sum_{V \in \Delta P^e_{max}(X) \cup C(X) \cup E(X)} c(V)$ is the maximum possible value of $L^>_{max}(X)$. If $\sum_{X \in E(X)} c(X) < 0$ then $\sum_{V \in \Delta P^e_{max}(X) \cup C(X)} c(V)$ is the maximum possible value for $L^<_{max}(X)$, otherwise $L^>_{max}(X)$ is the maximum possible value for $L^<_{max}(X)$.*

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]. □

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

4.2 Relating the Envelopes

In this section, we formally establish the relationship between the intervals over which the FBC and E_t bounds hold.

Since the envelope is the resource bound as a function of time, we should be able to determine the interval of time over which the event based bounds hold; then the maximum (minimum) envelope at t is the upper (lower) bound of all of the event based bounds that hold at t. 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 $\forall V \in P^{<C}_{max}(X)\{X \leq V\}$ to get a new RTN $\mathcal{R}'$. Note that the new pending set $P'(X) = \emptyset$ and the new closed set $C'(X) = P^<_{max}(X)$. Note also that $L^<_{max}(X)$ is the resource availability no earlier than $V^*_{ub'} = \max_{V \in C'(X)} V_{ub'}$. 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 $[V^*_{ub'}, X_{ub'}]$.

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* $V^*_{ub'} \leq t \leq X_{ub'}$.

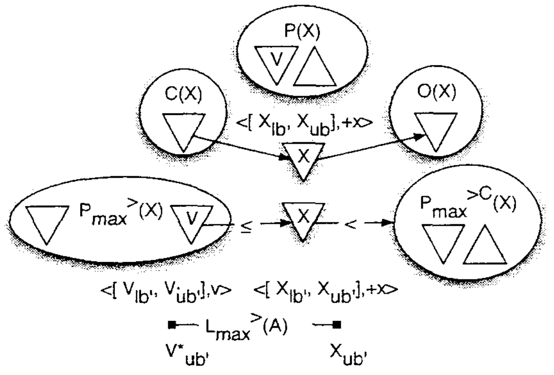
Proof. Since we add new constraints, $\forall t L_{max'}(t) \leq L_{max}(t)$. Since $L^{<}_{max}(X) = L_{max'}(t)$ we're done. □

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 $\forall V \in P^{>C}_{max}(X)\{X < V\}$ to get a new RTN $\mathcal{R}'$. Note that the new pending set $P'(X) = \emptyset$ and the new closed set $O'(X) = P^{>}_{max}(X)$. Note also that $L^{>}_{max}(X)$ is the resource availability no later than $W^{*}_{lb'} = \min_{W \in C'(X)} W_{lb'}$. Due to the new constraints X may have a new lower bound $X_{lb'}$, which defines the latest time that $L^{>}_{max}(X)$ holds. Finally, note that $L^{>}_{max}(X) = L_{max'}(t)$ over the interval $[X_{lb'}, W^{*}_{lb'}]$. An example is shown in Figure 5.

Theorem 4. *$L^{>}_{max}(X) \leq L_{max}(t)$ for $X_{lb'} \leq t \leq W^{*}_{lb'}$.*

Proof. Identical to the previous proof. □

(a) Resource Temporal Network decomposed relative to event X.



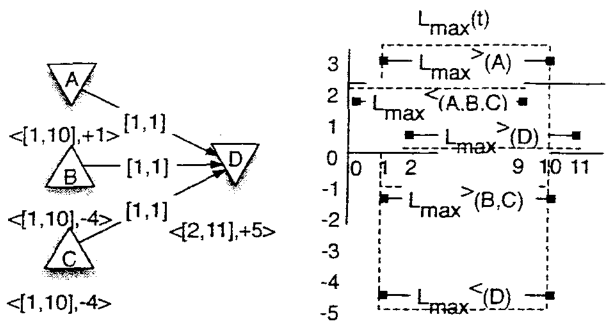
(b) The new Resource Temporal Network. The bound v* is the latest time of any event in $P_{max}{}^{>}(X)$. $L_{max}{}^{>}(A)$ holds over the interval $[V^{*}_{ub'}, X_{ub'}]$.

Fig. 5. Deriving the time intervals over which resource availability bounds hold.

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 primarily used to detect whether a scheduling algorithm can halt or must continue. In order to motivate our definition of dominance,



(a) Resource Temporal Network (b) Intervals over which event bounds hold; the Envelope is the maximum over all bounds that hold at any time t.

Fig. 6. An RTN for which E_t fails to detect that the problem is unsolvable.

suppose that some resource bounding procedure A is used in a chronological search framework with a fixed variable and value ordering heuristic. Then we would like A to domiante 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 domiante 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.*

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. □

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 □

Theorem 7. $FBC \prec E_t$

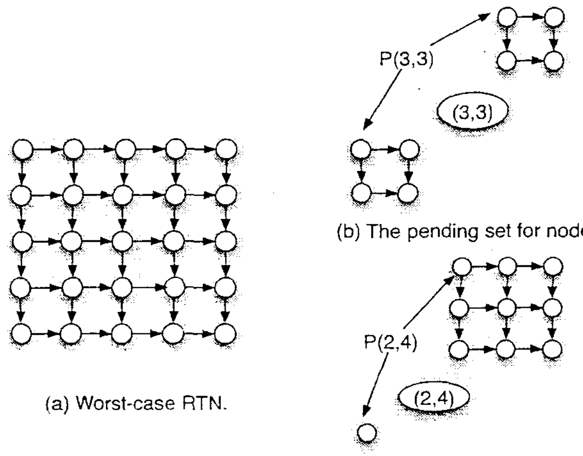
Proof. Theorem 6 shows that there is no RTN S for which $E_t \prec_S FBC$. The example in Figure 3 shows that there is at least one RTN R for which $FBC \prec_R E_t$. This completes the proof. □

6 Complexity

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 $O(n * MaxFlow(n, m))$, 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 $P(A) = P(B) = P(C) = P(D)$. 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 $P(X)$, leading to smaller flow problems. These observations suggest it should be possible to analyze the structure of the anti-precedence graph and reduce the number of flow problems to solve, with a resulting tighter bound on the complexity of calculating FBC.

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.



(a) Worst-case RTN.

(b) The pending set for node (3,3).

(c) The pending set for node (2,4).

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):

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

The first integral with respect to i is trivial. Substituting for $\sqrt{n}-i$, we see that $\int_{i=0}^{\sqrt{n}-1}(\sqrt{n}-i)di = (\sqrt{n}-i)^{2.5}\left(\frac{i-\sqrt{n}}{3.5}\right)$, so ultimately we get

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

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 $\Omega(n * MaxFlow(n,m))$.

7 Incrementally Calculating FBC

As stated previously, the incremental technique described in [7] shaves a factor of n off the cost of calculating E_t. This provides some hope that we can find a way to eliminate the factor of n "extra" cost for calculating FBC. A naive approach to employing the results in [7] for calculating FBC is inadequate to eliminate a factor of n from the complexity. 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 for which the incremental algorithm can be used to calculate FBC and reuse the previous flow is of length $2\sqrt{n}$ (either the diagonal or two edges). Each of these induces a total complexity of $O(MaxFlow(n,m))$ because, eventually, all events and arcs in the RTN will be considered in some flow problem. There are $O(\sqrt{n})$ such chains; once one chain is done, all the information about the previous flow must be reset, because at least one event that was previously closed becomes pending or open. The total complexity is therefore $O(\sqrt{n}MaxFlow(n,m))$. At worst, no ordering of the events can be found to eliminate this; at best non-trivial analysis of the precedence graph is necessary to find such an ordering.

A solution to this problem involves using the topological structure of the precedence graph to order the calls to Bound(X) and to *cache* the information from each flow problem that is solved. As long as $X_{i-1} \in C(X_i)$ no event in $C(Xi-1)$ ever enters $P(X_i)$ and no event in $P(X_{i-1})$ ever enters $O(X_i)$; this ensures that the incremental approach of [7] can be used. On the other hand, whenever X_i does not strictly follow X_{i-1}, we have cached the necessary data from the flow problem for some event Y that strictly preceded X_i, again ensuring that the incremental approach of [7] can be used. The node order requires beginning with a node with no predecessors. Implicitly the node order traverses the precedence graph using a topological sort. It's trivial to initialize the set of events with no precedences, and also trivial to mark each visited node to ensure a topological order is obeyed.

For the analysis of the algorithm, we simply fix an arbitrary topological order and concentrate on describing the storage of the flow information and the effective size of the largest flow problem that could be solved. We call the resulting algorithm $FBC - DFS$, and it is described in Figure 8 below.

```
FBC-DFS($\mathcal{R}, X_i$)
    Make the STN $S$ arc-consistent.
    Infer all anti-precedence and precedence links. (transitive closure)
    Collect all sets $P(X), C(X)$.
    Collect set $\mathcal{Y}$ of events with no predecessors
    $F_{max} = F_{min} = \emptyset$
    for each event $X_i$
        if $\neg(X_{i-1} \in C(X_i))$
            Find the latest $Y$ previously visited such that $Y \in C(X_i)$
            Pop $F_{max}$ and $F_{min}$ to $Y$
        end if
        Build flow problems for upper and lower bounds from $P(X), top(F_{max})$ and $top(F_{min})$
        Bound($X_i$)
        Push the flow solutions and events they correspond to onto $F_{max}$ and $F_{min}$
    end for
end
```

Fig. 8. A sketch of the Flow Balance Constraint implemented by using a Depth-First search through the precedence graph.

Theorem 8. *Let $\mathcal{R}$ be an RTN with m induced anti-precedence constraints in its STN S. Algorithm $FBC - DFS$ takes $O(MaxFlow(n, m))$ time and $O(nm)$ space.*

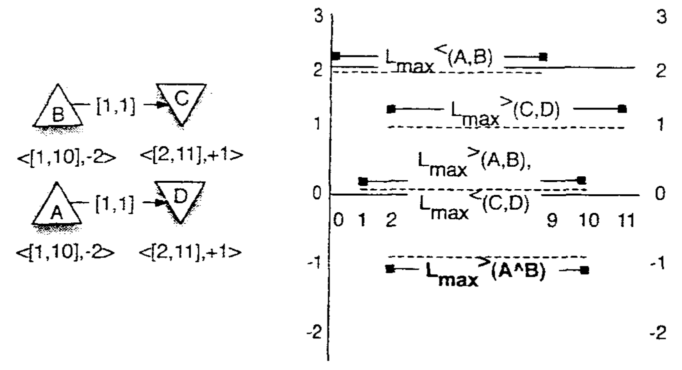
Proof. Ensuring events are searched in topological order imposes no significant overhead, nor does finding the correct event to back up to. If X_i strictly precedes X in the order that Bound(X_i) is invoked, then the incremental complexity argument of [7] applies. Essentially, we must (eventually) construct the flow problem for the entire STN.

At worst, each time we call Bound(X_i) we must store the resulting flow network. Because we are searching the graph depth-first, at worst we do so $O(n)$ times. If not, we must pop the stacks F_{max} and F_{min} each time we "back up". At worst we back up $O(n)$ times and store data on each of m flow edges. Only a constant amount of data is required for each edge. This cost is $O(mn)$, which is dominated by the complexity of the flow (for FIFO preflow-push, anyway, because $\sqrt{m} \leq n$). □

Note that weaker conditions on the events may lead to improved computational performance, but the conditions we have imposed are sufficient to produce an algorithm with the desired run-time complexity.

8 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 occured 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 violated.



(a) Resource Temporal Network (b) 1st and 2d order Flow Balance Constraint upper bounds. The second order bound $L_{max}^{>}$(A^B) proves the NTC is violated.

Fig. 9. An RTN for which FBC fails to disprove .

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 $P(X) \cap P(Y)$ to find the maximum availability strictly before both X and Y. We call this set of events $P^{<}_{max}(\neg(X \wedge Y))$. The second requires adding the constraint $X < Y$ and then solving the flow problem defined by $P(Y)$ and find the maximum availability strictly before Y. Call this set $P^{<}_{max}(X < Y)$. The last requires adding the constraint $Y < X$ and then solving the flow problem defined by $P(X)$ and find the maximum availability strictly before X. Call this set of events $P^{<}_{max}(Y < X)$. We define $L^{<}_{max}(X \wedge Y)$ as

$$\max\Big(\sum_{V \in P^{<}_{max}(\neg(X \wedge Y))} c(V),\quad \sum_{V \in P^{<}_{max}(X<Y)} c(V),\quad \sum_{V \in P^{<}_{max}(X<Y)} c(V) \qquad (1)$$

For $L^{>}_{max}(X \wedge Y)$ do the following. First, we solve the flow problem over the events of $P(X) \cup P(Y)$. Call the resulting set $P^{>}_{max}((X * Y))$. Now we must also include events in $E(X) \cup E(Y)$, but some of these may already be in $P(X) \cup P(Y)$, so these events must be discarded. Define $R(X \vee Y) = (E(X) \cup E(Y)) \cap (P(X) \cup P(Y)) \cup P^{>}_{max}((X * Y))$. We thus define $L^{>}_{max}((X \vee Y)) = \sum_{V \in R(X \vee Y)} c(V)$. 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 $P(A) \cap P(B) = \emptyset$. 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 occured. 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 3 flow problems must be solved per bound.

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. This is because $P(X \wedge Y) = P(X) \cup P(Y)$. 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).

9 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 theoretical 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. Furthermore, despite their apparent complexity, higher order variants of FBC may also prove worthwhile. The DFS-FBC algorithm could benefit from judicious node orderings. One possible reason to order the flow problems is to reduce total flow problem solving costs, another (possibly conflicting) goal is to reduce storage costs. Similarly, the higher-level consistency algorithms can be crafted to maximally exploit the incremental envelopes approach as well as judicious ordering of the sets of k events that must be contemplated to enforce the bounds.

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.

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.

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 is complicated by several factors. The pushable flow cannot be easily analyzed, but it is tempting to try assessing the relationship between the *number* and *size* of the flow problems to be solved during calculation of FBC by looking at the precedence and anti-precedence graphs. Appealing to graph theory may lead to both improved complexity analysis and, possibly, better algorithms as well.

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)