Tight Envelope Propagation for Producers and Consumers of Reservoirs

Armin Wolf

Fraunhofer FIRST, Kekuléstr. 7, D-12489 Berlin, Germany
armin.wolf@first.fraunhofer.de

Abstract. This paper addresses constraint-based scheduling of production and consumption events increasing respectively decreasing the inventory level of a common reservoir. A new incremental propagation algorithm is presented based on the envelope functions presented by KUMAR strongly bounding the level of a reservoir. For runtime examinations the algorithm as well as the balance constraint presented by LABORIE and time tabling for reservoirs are prototypically implemented and integrated in our Java constraint solving library `firstCS`. The examinations are based on the benchmarks generated by NEUMANN and SCHWINDT and on the heuristic search invented by LABORIE. The results show that our algorithm in combination with the balance constraint outperforms the balance constraint suppported by time tabling.

1 Introduction

Resource-constrained scheduling is still in the focus of current research, documented by the relevant conferences, like CP or CP-AI-OR. One reason may be the challenge to solve problem which are in general NP-complete in reasonable time, another the practical relevance for complex real world scheduling problems.

The termination of producers or consumers accessing reservoirs with limited capacities is of strong practical relevance. It arises, for instance, in resource-constrained project scheduling as well as in production planning if resources like water, electric power or other consumables have to be considered. Due to the close relation between reservoirs and cumulative resources it is obvious to consider producers and consumers of a common reservoir as activities to be allocated on a cumulative resource [8]. If this Constraint Programming(CP) modeling approach is used to solve such scheduling problems the specific nature of the problem is lost offering potentials for more efficient and even stronger propagation algorithms. Such a "'more native"' CP approach is presented in [1]. There, capacitive reasoning concerning reservoirs is interleaved with temporal reasoning on the production and consumption events.

Profile Propagation as proposed in [1], is similar to time-tabling techniques known for cumulative resources. It considers optimistic and pessimistic profiles *at time-points*. These profiles are conservative approximations of the levels of the reservoirs at a specific time-point only considering the temporal relationship with respect to this time-point. *Order Propagation* as proposed in [1] considers

optimistic and pessimistic profiles of reservoir levels *at events*. These profiles are identical to the so-called "'first order"' approximations of the reservoir levels presented in [5]. Order Propagation is more powerful than *Profile Propagation*: it deduces new precedence relations between events. In [5] some progress is made: the propagation techniques presented there are stronger than *Order Propagation*. Further in [5] the computed upper and lower bounds on the reservoir level are used while searching for a solution: either for backtracking in the case of a level underflow or overflow or for a successful termination of the search when all events are *safe* , i.e. the computed bounds are always inside the range of allowed reservoir levels. Consequently, MUSCETTOLA reasons in [6] that "'bound tightness is extremely important computationally since both as backtracking and termination criteria it can save a potentially exponential amount of search when compared to a looser bound."' Unfortunately, neither in [1] nor in [5] conflicting producers and consumers that can be scheduled at or just before/after an event are considered in their bound computations. However, conflicting events are considered in [4,6] for tight bounds computations. Both presented approaches reduces the problem of finding events that have the optimal contribution to the reservoir envelope to maximum flow problems. While the focus in [6] lies on a feasibility, i.e. effective and efficient algorithms in general, the focus in [4] lies on incremental computation, i.e. the reuse of computation while updating the envelopes in the case of any change: during search the envelopes must be re-computed at every choice-point, so even small improvements save potentially an exponential amount of work.

Before presenting the propagation technique that integrates an adaptation of incremental envelope computation in [4] with the pruning similar to *Profile Propagation* the considered entities, i.e. events, constraints and scheduling problems are defined. Then, the used data structures, i.e. distance graphs, flow graphs and their residuals as well as the according algorithms are introduced and related to the considered scheduling problems. Last but not least the results of some runtime examinations are presented.

2 Entities and Notations

2.1 Production and Consumption Events

In the considered scheduling model there are production and consumption events changing the inventory level of their common reservoir over the time. Formally the events are defined as follows:

Definition 1 (Production and Consumption Events).
An event e *is defined by its* time point $t(e)$ *where this events occurs. In general, the value of this time point is not determined, however, it is restricted to a* finite domain *ranging from its* earliest time $t_{\min}(e)$ *to its* latest time $t_{\max}(e)$*, i.e.*

$$t(e) \in [t_{\min}(e), t_{\max}(e)] \subset \mathbb{Z} \text{ where } t_{\min}(e) \leq t_{\max}(e) \text{ holds.}$$

A production event p *is an event additionally defined by the* quantity $q(p)$ *of* produced amounts*. In general, the value of this quantity is not determined, however, it is restricted to a* finite domain *ranging from its* minimal quantity $q_{\min}(p)$ *to its* maximal quantity $q_{\max}(p)$*, i.e.*

$$q(p) \in [q_{\min}(p), q_{\max}(p)] \subset \mathbb{Z} \text{ where } 0 < q_{\min}(p) \leq q_{\max}(p) \text{ holds.}$$

A consumption event c *is similar to an production event; the only difference is that its (consumed)* quantity $q(c)$ *is negative, i.e.*

$$q(c) \in [q_{\min}(c), q_{\max}(c)] \subset \mathbb{Z} \text{ where } q_{\min}(c) \leq q_{\max}(p) < 0 \text{ holds.}$$

A *neutral* event n is an event with a *quantity* $q(n) = 0$. It will not change the load of an reservoir, however, such events might occur in temporal constraints, defined as follows.

2.2 Temporal Constraints

In our considered scheduling model the temporal relationship of events is represented as a *Simple Temporal Problem* according to [2]:

Definition 2 (Temporal Constraints). *Let a set of events E be given. The* temporal constraints $T(E)$ *on E are constituted by a set of of inequalities $t(e) - t(e') \leq d$ where d is an integer constant and e, e' are in E.*

It should be noted that any two events $e, e' \in E$ *sharing* the same time point satisfy the temporal constraints $t(e) - t(e') \leq 0$ and $t(e') - t(e) \leq 0$.

2.3 Reservoirs

A reservoir is is a multi-capacity resource with limited inventory level that will be decreased by consumption events and increased by production events over the time. A reservoir has a limited maximal capacity and a minimal capacity which is sometimes different from zero, e.g. if for safety reasons, there must be an "iron reserve":

Definition 3 (Reservoirs). *A* reservoir R *is defined by its minimal capacity $R_{\min} \in \mathbb{Z}$, its maximal capacity $R_{\max} \in \mathbb{Z}$ and its initial (inventory) level $Q \in \mathbb{Z}$.*

Further let P be a finite set of production events and C be a finite set of consumption events accessing R*. Then the* reservoir constraint *is satisfied for R and it accessing events $P \cup C$ if the inequalities*

$$\forall \tau : R_{\min} \leq Q + \sum_{x \in P \cup C | t(x) \leq \tau} q(x) \leq R_{\max}$$

are satisfied. Here the function

$$l(\tau) := Q + \sum_{x \in P \cup C | t(x) \leq \tau} q(x)$$

is called the level *(function) of the reservoir changing over the time.*

Obviously, the reservoir constraint implies that $R_{\min} \leq Q \leq R_{\max}$ must hold. Sometimes the minimal capacity is normalized to zero by subducting $R_{\min}$ from Q and $R_{\max}$: $R'_{\min} = 0, Q' = Q - R_{\min}, R'_{\max} = R_{\max} - R_{\min}$.

Further, it should be noted that temporal constraints between the production and/or consumption events are not considered by the reservoir constraint. In the following definition of reservoir scheduling problems they will be taken into account, too.

2.4 Reservoir Scheduling Problems

A reservoir scheduling problem considers the reservoir constraint as well as the temporal constraints on the production and consumption events accessing a common reservoir:

Definition 4 (Reservoir Scheduling Problems and their Solutions). *A* reservoir scheduling problem *is defined by a reservoir R and its accessing production and consumption events $P \cup C$ (cf. Def. 3), the temporal constraints on $P \cup C$ as well as all other constraints restricting the time points and capacities, in particular the reservoir constraint for R and it accessing events $P \cup C$.*

A solution *or* schedule *S of a reservoir scheduling problem is a valuation (or a* labeling*) of the time points $t(x) = t_S(x), t_S(x) \in \mathbb{Z}$ and of the quantities $q(x) = q_S(x), q_S(x) \in \mathbb{Z}$ for all events $x \in P \cup C$ such that all constraints are satisfied.*

The restriction to one reservoir in the considered scheduling problems is not crucial. In order to solve scheduling problems with several reservoirs, the subproblems for each reservoir has to be solved additionally satisfying the temporal constraints on events with different reservoirs. In CP this is rather straightforward due to constraint propagation. So in the rest of the paper – except for runtime examinations (cf. Sec. 4) – we focus on reservoir scheduling problems for *one* single reservoir.

3 Tight Envelope Propagation

In [4] an approach is presented to compute a tight *upper envelope function* g such that $l(\tau) \leq g(\tau)$ holds as well as a tight *lower envelope function* h such that $h(\tau) \leq l(\tau)$ holds:

$$g(\tau) = Q + \max_{S \text{ is a temporal consistent schedule}} \sum_{x \in P \cup C | t_S(x) \leq \tau} q_S(x) \text{ and}$$

$$h(\tau) = Q + \min_{S \text{ is a temporal consistent schedule}} \sum_{x \in P \cup C | t_S(x) \leq \tau} q_S(x) \ .$$

where the notion *temporal consistent schedule* means that for each event $x \in P \cup C$, there is an assignment $t(x) = t_S(x)$ with $t_S(x) \in [t_{\min}(x), t_{\max}(x)]$ $q(x) = q_S(x)$ with $q_S(x) \in [q_{\min}(x), q_{\max}(x)]$ such that the temporal constraints $T(P \cup C)$ are satisfied. However the reservoir constraint is not considered

here. Obviously, these functions define indeed a tight *envelope* of l, because the set of temporal consistent schedules is a superset of the set of schedules additionally satisfying the reservoir constraint.

In the following it is shown how the computation of the upper envelope function g will be performed in the context of propagation in (finite domain) CP adding some appropriate pruning of the search space of potential schedules restricting the time windows and the quantities of the events. Due to the symmetry between g and h (cf. [4]) the considerations concerning the lower envelope function h are omitted.

3.1 Distance Graphs

For the propagation of the temporal constraints $T(P \cup C)$ and the restrictions on the time points $t(x) \in [t_{\min}(x), t_{\max}(x)]$ for each event $x \in P \cup C$ a distance graph is used as proposed in [4].

Definition 5. *A* distance graph $D(V, E)$ *consists of a finite set of* vertexes V *and a finite set of directed weighted* edges E. *Each edge $e \in E$ is defined by a pair of annotated vertexes $e = (u, v)_w \in V \times V \times \mathbb{Z}$ where w is the* weight *of the edge from u to v. By convention it holds $(u, u)_0 \in E$ for each vertex $u \in V$.*

There is a negative cycle *in the distance graph if there are edges $(v_{i-1}, v_i)_{w_i} \in E$ for $i = 1, \ldots, n$ such that $v_n = v_0$ and $\sum_{i=1}^{n} w_i < 0$ hold.*

It is well-known that the shortest paths between all pairs of vertexes $u, v \in E$ are well-defined if the distance graph has no negative cycle. In this case there are well known all-pair shortest paths algorithms, e.g. the so-called Floyd-Warshall algorithm or the Bellman-Ford algorithm, computing the minimal distances between any two vertexes in a distance graph. Given a distance graph $D(V, E)$ the minimal distance function $\mathsf{dist} : V \times V \to \mathbb{Z} \cup \{\infty\}$ is defined recursively by

$$\mathsf{dist}(u, v) = \begin{cases} 0 & \text{if } u = v, \\ \min_{x \in V}(\mathsf{dist}(u, x) + \mathsf{weight}(x, v)) & \text{otherwise,} \end{cases}$$

where

$$\mathsf{weight}(u, v) = \begin{cases} w \text{ if } (u, v)_w \in E, \\ \infty \text{ otherwise.} \end{cases}$$

In the following, the distance graph $D(P \cup C) = D(V, E)$ is considered, where the vertexes in V are the time points $t(x)$ of the events $x \in P \cup C$ respective 0 and where the edges are

$$\begin{aligned} E = &\{(t(x), t(y))_d \mid t(y) - t(x) \leq d \in T(P \cup C)\} \\ &\cup \{(0, t(x))_{t_{\max}(x)} \mid x \in P \cup C\} \cup \{(t(x), 0)_{-t_{\min}(x)} \mid x \in P \cup C\} \ . \end{aligned}$$

Thus a temporal consistent schedule exists for $P \cup C$ if the distance graph $D(P \cup C)$ has no negative cycle (cf. [4, Lemma 1]).

The distance graph $D(P \cup C)$ will be used for adjusting the temporal domains of the events: For each event $x \in P \cup C$ it holds

$$-\mathsf{dist}(t(x), 0) \leq t(x) \leq \mathsf{dist}(0, t(x)) \tag{1}$$

restricting the bounds of its time interval updated accordingly:

$$t'_{\min}(x) = \max(t_{\min}(x), -\mathsf{dist}(t(x), 0) \text{ and } t'_{\max}(x) = \min(t_{\max}(x), \mathsf{dist}(0, t(x)) \ .$$

This propagation is obviously correct and idempotent, i.e. computes a fix-point, because the distance function computes all minimal distances.

3.2 Active Events, Conflicts and Conflict Sets

Definition 6 (P-Active Event, cf. [4, Definition 1]). *An event is said to be* p-active *if it contributes its maximum to the total production by time t. This means that a production event $p \in P$ is* p-active *at time t if it is scheduled before or at t – i.e. $t(p) \leq t$ – and a consumption event $c \in C$ is* p-active *at time t if it is scheduled after t – i.e. $t(c) > t$.*

Lemma 1 (cf. [4, Lemma 2]). *A schedule that achieves $g(t)$ at time t maximizes $\sum_{x \in P \cup C | x \text{ is p-active at } t} q_{\max}(x)$.*

Lemma 2 (adopted from [4, Lemma 3]). *A production event $p \in P$ can be made* p-active *at time t if $t_{\min}(p) \leq t$ holds after updating $t_{\min}(p)$ according to (1). Analogously, a consumption event $c \in C$ can be made* p-active *at time t if $t_{\max}(c) > t$ holds after updating $t_{\max}(c)$ according to (1).*

Definition 7 ((Minimal) Conflict, cf. [4, Definition 2]). *A* conflict *is a set of events which cannot be made* p-active *simultaneously at a given time t. A* minimal conflict *is a conflict having no proper subset which is also a conflict, i.e. it is minimal by set inclusion.*

This means that a set of events can be simultaneously made *p-active* at time t if there is no subset constituting a (minimal) conflict. Thus the size of a minimal conflict is 1 or 2 (cf. [4, Lemma 5]). Further, the following property holds:

Lemma 3. *A minimal size-2 conflict consists of a production and a consumption event.*

Proof. Let a minimal size-2 conflict at time t be given. Further, it is assumed that this conflict consists of two production events $p_1 \neq p_2$, i.e. $t_{\min}(p_1) > t$ and $t_{\min}(p_2) > t$. Then, either production event constitute a size-1 conflict, i.e. the size-2 conflict is not minimal. The assumption that the conflict set consists of two consumption events results in an analogous contradiction. Thus the minimal size-2 conflict consists of a production and a consumption event. □

Lemma 4. *A consumption event $c \in C$ and a production event $p \in P$ constitute a size-2 conflict if $\mathsf{dist}(p, c) \leq 0$ holds. Conversely, $\mathsf{dist}(p, c) \leq 0$ holds if p and c constitutes a minimal size-2 conflict for each $t \in [t_{\min}(p), t_{max}(c) - 1]$.*

Proof. $\mathsf{dist}(p,c) \leq 0$ means that $t(c) \leq t(p)$ holds, i.e. c will not be scheduled after p which implies that both cannot be made p-active simultaneously at any t.

It is assumed that $\mathsf{dist}(p,c) > 0$ holds. Then for each $t \in [t_{\min}(p), t_{max}(c)-1]$ there will be a schedule with $t(p) \leq t < t(c)$, such that p and c does not constitute any conflict.

Corollary 1 (Detecting minimal size-2 conflicts). *A production event $p \in P$ and a consumption event $c \in C$ constitute a minimal size-2 conflict at time t, if and only if*

- $\mathsf{dist}(p,c) \leq 0$ *holds and*
- $t \in [t_{\min}(p), t_{max}(c)-1]$ *holds after updating* $t_{\min}(p), t_{max}(c)$ *according to (1).*

If there is a minimal size-2 conflict between a production event p and a consumption event c at time t then it is impossible to schedule p at time t without scheduling c before or at least at t, i.e. if p contributes (positive) to the level of the reservoir then c contributes (negative), too.

Concerning the computation of g one is in particular interested in the production events of all minimal size-2 conflicts at time t such that their (positive) contributions to the level function is minimally compensated by the (negative) contributions of their conflicting consumption events:

Definition 8 (Minimal Conflict Set, Maximally Closed). *Let*

$$K(t) = \{(p,c) \mid p \in P, c \in C, \{p,c\} \text{ constitute a minimal size-2 conflict at } t\}$$

be the minimal conflict set *at t. A subset $M(t) \subseteq K(t)$ is* closed (under activation) *if $\{(x,z) \mid (x,z) \in K(t)$ and there is a $(x,y) \in M(t)\} \subseteq M(t)$ holds. A closed subset $M(t) \subseteq K(t)$ is* maximally closed *if*

$$\sum_{(x,y)\in M(t)} q_{\max}(x) + q_{\max}(y) \geq \sum_{(x,y)\in N(t)} q_{\max}(x) + q_{\max}(y)$$

holds for each closed subset $N(t) \subseteq K(t)$.

Due to the characteristics of maximally closed subsets of minimal conflict sets, the following theorem holds obviously:

Theorem 1. *Let a maximally closed subset $M(t)$ of the minimal conflict set at t be given.*[1] *Then it holds*

$$g(t) = Q + \sum_{x\in P(t)\cup C(t)} q_{\max}(x) + \sum_{(x,y)\in M(t)} q_{\max}(x) + q_{\max}(y) \ ,$$

where $P(t) \cup C(t)$ contains the production events that can be made p-active at t and the consumption events that cannot be made p-active at t neither involved in a minimal size-2 conflict at t:

$$P(t) = \{p \mid p \in P, t_{\min}(p) \leq t \text{ and } (p,c) \notin K(t) \text{ for any } c \in C\} \tag{2}$$

$$C(t) = \{c \mid c \in C, t_{\max}(c) \leq t \text{ and } (p,c) \notin K(t) \text{ for any } p \in P\} \ . \tag{3}$$

[1] In general maximally closed subsets are not uniquely determined.

3.3 Maximum Flow for Maximally Closed Conflict Sets

KUMAR and MUSCETTOLA [4,6] proposed to use maximum flow calculation in flow graphs for the computation of a maximally closed, minimal conflict set $M(t)$. In the following their approach is adapted and refined. In particular, combined with propagation an alternative incremental update of the considered flow graphs and their maximum flows is presented.

Definition 9 (Flow Graph of a Minimal Conflict Set). *Let a minimal conflict set $K(t)$ for time t be given. The according* flow graph $G(K(t)) = G_t = \langle V_t, E_t, cap_t \rangle$ *is defined by*

- *a* capacity function $cap_t : E_t \to \mathbb{N}$
- *a set of vertexes V_t consisting of a* source S*, a* target T *and all productions and consumption events in $K(t)$, i.e.*

$$V_t = \{S, T\} \cup \{p \mid (p, c) \in K(t)\} \cup \{c \mid (p, c) \in K(t)\} \ ,$$

- *a set of directed edges E_t consisting of*

$$\begin{aligned} S \to p \text{ where } cap_t(S \to p) &= \ q_{\max}(p) \ \text{ for each } (p, c) \in K(t), \\ p \to c \text{ where } cap_t(p \to c) &= \quad \infty \quad \ \text{ for each } (p, c) \in K(t), \\ c \to T \text{ where } cap_t(c \to T) &= -q_{\max}(c) \text{ for each } (p, c) \in K(t). \end{aligned}$$

Considering the flow graph of a minimal conflict set the maximal closed conflicts coincides with a set that is based on a maximum flow in such flow graphs:

Definition 10 (Maximum Flows). *Let a flow graph G_t be given. A function $f_t : E_t \to \mathbb{N}$ is a* flow *in G_t (subject to cap_t) if the* flow condition $\sum_{u \to v \in E_t} f_t(u \to v) = \sum_{v \to w \in E_t} f_t(v \to w)$ *holds for each vertex $v \in V_t \setminus \{S, T\}$ and the* capacity condition $f_t(e) \leq cap_t(e)$ *holds for each $e \in E_t$.*

The value *of a flow f_t – denoted $|f_t|$ – is*[2]

$$|f_t| = \sum_{S \to p \in E_t} f_t(S \to p) = \sum_{c \to T \in E_t} f_t(c \to T) \ .$$

A flow f_t is maximal *if $|f_t| \geq |g_t|$ holds for each flow g_t in G_t.*

Definition 11 (Residual Graphs and Reachables). *Let a flow graph G_t and a flow function f_t be given. The* residual capacity $rescap_{f_t}$ *is defined by*

$$rescap_{f_t}(u \to v) = \begin{cases} cap_t(u \to v) - f_t(u \to v) & \text{if } u \to v \in E_t, \\ f_t(v \to u) & \text{if } v \to u \in E_t, \\ 0 & \text{otherwise.} \end{cases}$$

The residual graph *of G_t with respect to f_t is the graph $R_{f_t} = \langle V_t, E_t \cup \{u \to v \mid v \to u \in E_t\}, rescap_{f_t} \rangle$. Further, let*

$$M_{f_t} = \{e \mid e \in V(t) \setminus \{S, T\} \text{ and } e \text{ is reachable from } S \text{ in } R_{f_t}\}$$

be the (set of) vertexes reachable from S *in the residual graph.*

[2] The definition given here is specialized according to the structure of the *bipartite* flow graphs considered here.

Summarizing the main result of KUMAR [4] it holds

$$g(t) = \sum_{p \in P(t)} q_{\max}(p) + \sum_{c \in C(t)} q_{\max}(p) + \sum_{e \in M_{f_t}} q_{\max}(e)$$

for each maximum flow f_t. Due to the considerations made, it is obvious that $g(t)$ is a piecewise constant function that changes its value at most in the following cases:

1. $t = t_{\min}(p)$ for each $p \in P$
   (a) either increasing $K(t)$ if there are minimal size-2 conflicts (p, c),
   (b) or increasing $P(t)$ otherwise.
2. $t = t_{\max}(c)$ for each $c \in C$
   (a) possibly decreasing $K(t)$
   (b) and possibly increasing $C(t)$.

In the case 1a new minimal size-2 conflicts have to be added to the residual graph of the minimal conflict set. This is performed by Alg. 1. It is important that the augmentation of a residual graph by the use of Alg. 1 is well-defined. In particular, the residual function as well as the (implicitly augmented) flow function are still well-defined.

Algorithm 1: Adds a new minimal size-2 conflict in a residual graph

Input: a minimal size-2 conflict (p, c) consisting of a production event p and a consumption event c; a residual graph $\langle V, E, rescap \rangle$ defined by its vertexes V, its edges E and its residual capacity $rescap$.

Output: an augmented, well-defined residual graph.

```
1  addConflict((p, c), ⟨V, E, rescap⟩) ≡
2  begin
       // It is assumed that p ∉ V holds.
3      V := V ∪ {p};
4      E := E ∪ {S → p, p → S};
5      rescap(S → p) := q_max(p);
6      rescap(p → S) := 0; // implying f(S → p) = 0
7      E := E ∪ {p → c, c → p};
8      rescap(p → c) := ∞;
9      rescap(c → p) := 0; // implying f(p → c) = 0
10     if c ∉ V then
11         V := V ∪ {c};
12         E := E ∪ {c → T, T → c};
13         rescap(c → T) := −q_max(c);
14         rescap(T → c) := 0; // implying f(c → T) = 0
15     return ⟨V, E, rescap⟩;
16 end
```

In the case 2a already added minimal size-2 conflicts have to be removed from the residual graph of the minimal conflict set. This is performed by Alg. 2.

It is important that the reduction of a residual graph by the use of Alg. 2 is well-defined. In particular, the residual function as well as the (implicitly adapted) flow function are still well-defined.

Any change of a residual graph with maximum flow, adding or removing minimal size-2 conflicts by the use of Alg. 1 or Alg. 2 will indeed result in a graph with an admissible flow which is in general not maximal. Thus the flow can be increased incrementally to a maximal flow using Alg. 3 – an incarnation of the so-called FORD-FULKERSON algorithm – by *path augmentation*: any search for a path from S to T in the residual graph such that the flow along this path can be augmented.[3] Consequenty, findAugmentingPath searches along the edges in the given residual graph and returns an acyclic path Π and a integer value δ. If there is not any graph that can be augmented, an empty sequence, is returned. Otherwise, a sequence of the vertexes along the path and a positive value δ are returned. The value δ is the delta to be added to the flow along the returned path.

Algorithm 2: Removes a size-2 conflict in a residual graph

Input: a minimal size-2 conflict (p, c) consisting of a production event p and a consumption event c; a residual graph $\langle V, E, rescap\rangle$ defined by its vertexes V, its edges E and its residual capacity $rescap$.

Output: a reduced, well-defined residual graph.

```
1  removeConflict((p, c), ⟨V, E, rescap⟩) ≡
2  begin
3      let f := rescap(c → p);
4      rescap(S → p) := rescap(S → p) + f;
5      rescap(p → S) := rescap(p → S) − f;
6      rescap(c → T) := rescap(c → T) + f;
7      rescap(T → c) := rescap(T → c) − f;
8      E := E \ {p → c, c → p};
9      if there is no edge p → c' ∈ E for any c' ∈ C then
10         E := E \ {S → p, p → S};
11         V := V \ {p};
12     if there is no edge p' → c ∈ E for any p' ∈ P then
13         E := E \ {c → T, T → c};
14         V := V \ {c};
15     return ⟨V, E, rescap⟩;
16 end
```

Algorithm 4 incrementally computes the piecewise constant upper envelope function g for a given reservoir scheduling problem. Therefore, the input of the algorithm consists of a set of production events P and a set of consumption events C and a sequences of time points $t_1, \ldots, t_n$ sorted in ascending order,

[3] A depth-first search is applied in our implementation.

Algorithm 3: Computes the maximum flow in a residual graph

Input: a residual graph $\langle V, E, rescap\rangle$ defined by its vertexes V, its edges E and its residual capacity $rescap$.

Output: an updated residual graph with maximum flow.

```
1  computeMaxFlow(⟨V, E, rescap⟩) ≡
2  begin
3      (Π, δ) = findAugmentingPath(⟨V, E, rescap⟩);
4      while Π is not the empty path do
5          let Π = (e_0, e_1, ..., e_n); // where e_0 = S and e_n = T
6          let curr = S;
7          for i = 1, ..., n do
8              let next := e_i; // adjust flow residual capacities:
9              rescap(curr → next) := rescap(curr → next) − δ;
10             rescap(next → curr) := rescap(next → curr) + δ;
11             curr := next;
12         (Π, δ) = findAugmentingPath(⟨V, E, rescap⟩)
13     return ⟨V, E, rescap⟩;
14 end
```

i.e. $t_1 < t_2 < \cdots < t_n$ and containing all time bounds of the events in $P \cup C$, i.e. for each $x \in P \cup C$ and each $t_{\min}(x)$ and $t_{\max}(x)$ there are t_i, t_j with $1 \leq i, j \leq n$ such that $t_{\min}(x) = t_i$ and $t_{\max}(x) = t_j$ holds. It is important that these time bounds are up-to-date according to the distance graph $D = D(P \cup C)$ (cf. (1)) which will be extended by additional temporal constraints in the pruning phase of Alg. 4 performed by Alg. 5.

Algorithm 5 performs pruning of the time intervals of the events based on g at a given time t. Three different sets of events resulting from the envelope computation are considered:

- a set of active non-conflicting consumption events $N_{\min\leq}$,
- a set of non-conflicting contributing consumption events C_g,
- a set of non-conflicting contributing production events P_g.

While P_g corresponds to $P(t)$ in (2) and C_g corresponds to $C(t)$ in (3) the set of $N_{\min\leq}$ consists of consumption events which can be scheduled before, at or after t not yet contributing to g. First of all, Alg. 5 checks whether $g(t)$ is below the minimal capacity $R_{\min}$ of the reservoir; if so, the reservoir scheduling problem has no solution. Thus an inconsistency exception is thrown. Then the given sets are considered: Each consumption event $x \in N_{\min\leq}$ has to be scheduled after t if any earlier scheduling causes an underflow, i.e.$g(t)+q_{\max}(x)$ is below the minimal capacity $R_{\min}$ of the reservoir. Thus an edge representing the inequality $t(x) > t$ resp. $0 - t(x) \leq -(t+1)$ is added to the distance graph. The minimal quantity of each consumption event $x \in C_g$ has to be increased if it is contributing to g instead of its maximal quantity causes an underflow, i.e. $g(t)-q_{\max}(x)+q_{\min}(x)$ is below the minimal capacity $R_{\min}$ of the reservoir. Then $q_{\min}(x)$ will be increased

Algorithm 4: Propagation along the computed upper envelope function g.

Input: a set of production events P, a set of consumption events C, a sequence of increasing time points $t_1 < \cdots < t_n$ covering the minimal/maximal times of the events in $P \cup C$ and a distance graph $D = D(P \cup C)$.

Output: an extended and updated distance graph.

propagateAlongUpperEnvelope$(P, C, (t_1, \ldots, t_n), D) \equiv$
begin
  let $C_{\min\leq} := \emptyset$; `// the set of` *`active`* `consumption events`
  let $C_g := \emptyset$; `// the set of consumption events contributing to` g
  let $P_g := \emptyset$; `// the set of production events contributing to` g
  let $M := \emptyset$; `// the set of contributing conflicting events`
  let $K := \emptyset$; `// the set of minimal size-2 conflicts`
  let $V := \emptyset$; `// the set of vertexes of the residual graph` R
  let $E := \emptyset$; `// the set of edges of the residual graph` R
  let $rescap := \bot$; `// the residual capacity function of` R
  let $g := Q$; `// the (initial) value of the envelope function` g
  for $i = 1, \ldots, n$ **do**
    let isChanged := false; `// signals whether` R `has changed`
    foreach *consumption event* $c \in C$ *such that* $t_i = t_{\min}(c)$ **do**
      $C_{\min\leq} := C_{\min\leq} \cup \{c\}$;
    foreach *consumption event* $c \in C$ *such that* $t_i = t_{\max}(c)$ **do**
      $C_{\min\leq} := C_{\min\leq} \setminus \{c\}$;
      if $c \notin M$ **then** $g := g + q_{\max}(c)$;
      $M := M \setminus \{c\}$; $C_g := C_g \cup \{c\}$;
      foreach *production event* $p \in P$ *with* $(p, c) \in K$ **do**
        $K := K \setminus \{(p, c)\}$;
        $\langle V, E, rescap\rangle :=$ removeConflict$((p, c), \langle V, E, rescap\rangle)$;
        isChanged := true;
        if $p \notin V \wedge p \notin M$ **then** p is conflict-free and not yet contributing
          $g := g + q_{\max}(p)$;
        $M := M \setminus \{p\}$; $P_g := P_g \cup \{p\}$;
    foreach *production event* $p \in P$ *such that* $t_i = t_{\min}(p)$ **do**
      let hasConflict := false; `// signals whether` p `is in conflict`
      foreach *consumption event* $c \in C_{\min\leq}$ **do**
        if (p, c) *constitutes a minimal size-2 conflict* **then**
          $K := K \cup \{(p, c)\}$; `// add to the conflicts`
          $\langle V, E, rescap\rangle :=$ addConflict$((p, c), \langle V, E, rescap\rangle)$;
          hasConflict := true; isChanged := true;
      if $\neg$hasConflict **then** there was no minimal size-2 conflict
        $g := g + q_{\max}(p)$; $P_g := P_g \cup \{p\}$;
    if isChanged **then** the flow graph has changed; thus update
      let $R := \langle V, E, rescap\rangle :=$ computeMaxFlow$(\langle V, E, rescap\rangle)$;
      let $M' := \{e \mid e \in V \setminus \{S, T\}$ and e is reachable from S in $R\}$;
      $g := g - \sum_{d \in M \setminus M'} q_{\max}(d) + \sum_{e \in M' \setminus M} q_{\max}(e)$;
      $M := M'$;
    $D :=$ pruneOnUpperAt$(g, t_i, C_{\min\leq} \setminus V, C_g, P_g, D)$;
  update the time bounds for each $x \in P \cup C$ according to D (cf. (1));
  return D;
end

to $q'_{\min}(x)$ such that $g(t) - q_{\max}(x) + q'_{\min}(x) = R_{\min}$ holds. Each production event $x \in P_g$ has to be scheduled before or at t if any later scheduling causes an underflow, i.e. $g(t)+q_{\max}(x)$ is below the minimal capacity $R_{\min}$ of the reservoir. Thus an edge representing the inequality $t(x) \leq t$ resp. $t(x) - 0 \leq t$ is added to the distance graph. In this case further pruning is possible: The minimal quantity of x has to be increased if this minimal contribution to g causes an underflow, i.e. $g(t) - q_{\max}(x) + q_{\min}(x)$ is below the minimal capacity $R_{\min}$ of the reservoir. Then $q_{\min}(x)$ will be increased to $q'_{\min}(x)$ such that $g(t) - q_{\max}(x) + q'_{\min}(x) = R_{\min}$ holds.

It is assumed that any extension of the distance graph updates the minimal distances between the vertexes incrementally. Further it is assumed that during this updating negative cycles are detected and an inconsistency exception is thrown in the case of a detection.

Algorithm 4 has to be repeated until either an inconsistency exception is thrown or a fix-point is reached in order to perform maximal propagation.

Algorithms 4 and 5 are correct due to the fact that the computation of g is an incremental version of the computation presented in [4, Fig. 2]. Further, the pruning performed by Alg. 5 will obviously not exclude any solution of the scheduling problem. Termination of both algorithms is obvious.

Algorithm 5: Pruning at a time t based on the upper envelope function g.

Input: the value of an upper envelope function g at time t, a set of active non-conflicting consumption events $N_{\min\leq}$, two sets of contributing events C_g (consuming) and P_g (producing) and a distance graph D.

Output: an extended and updated distance graph.

```
1  pruneOnUpperAt(g, t, N_min≤, C_g, P_g, D) ≡
2  begin
3      if g < R_min then there is an obvious level underflow
4          throw inconsistencyException;
5      foreach c ∈ N_min≤ do
6          if g + q_max(c) < R_min then c scheduled before/at t causes an underflow
7              D := D ∪ {(t(c), 0)_−(t+1)}; // updates t(c) ≥ t + 1
8      foreach c ∈ C_g do
9          if g − q_max(c) + q_min(c) < R_min then q_min of c causes an underflow
10             q_min(c) := q_max(c) − g + R_min; // prunes the quantity
11     foreach p ∈ P_g do
12         if g − q_max(p) < R_min then
13             D := D ∪ {(0, t(p))_t}; // updates t(p) ≤ t
14             if g − q_max(p) + q_min(p) < R_min then underflow for q_min of p
15                 q_min(p) := q_max(c) − g + R_min; // prunes the quantity
16     return D;
17 end
```

4 Runtime Examinations

For runtime examinations Algs. 4 and 5 as well as the balance constraint presented in [5] and time tabling are prototypically implemented in our Java constraint solving library `firstCS` [3]. The examinations are based on the 90 benchmark problems of size 20 presented in [7]. There, 20 events, either consuming or producing inventory on some of 5 reservoirs, have to be scheduled within minimal time. In order to solve these optimization problems the heuristic search strategies presented in [5] are applied. If the initial constraint propagation detects no inconsistency it is tried to find a solution at the deduced lower bound of the total completion time. If this fails *dichotomic branch-and-bound* search is applied starting with the deduced lower bound and the upper bound proposed in [7]. Search stops either with an optimal schedule or without a solution. The performance of the balance constraint plus envelope propagation is compared to the balance constraint with time tabling (cf. [5]). Table 1 summarizes the results: in the first column the applied search strategies are listed, the following columns show the elapsed total runtime[4] and the performed backtracking steps (# BTs) for the compared approaches and the last columns shows the improvement due to envelope propagation (cf. Alg. 4) with respect to the number of backtracking steps. The net improvement (w.r.t. # BTs) varies between 85 % and 662 %.

Table 1. Results of the examinations performed on the benchmarks of size 20

search	balance + envelope		balance + time tabling		improvement
strategy	time (secs.)	# BTs	time (secs.)	# BTs	factor (# BTs) %
1, a	6.12	1046	14.72	7969	**762**
1, b	7.33	1950	16.35	8725	447
1, c	7.16	1931	16.94	8442	437
2, a	20.25	10870	27.28	20059	*185*
2, b	9.69	3193	23.80	15052	471
2, c	9.80	3047	18.42	10157	333
3, a	5.86	1046	14.80	7969	**762**
3, b	6.19	1366	14.52	7646	560
3, c	7.12	1936	16.19	8624	445

5 Conclusion, Current and Future Work

An incremental propagation algorithm for production and consumption events accessing a common reservoir is presented. The propagation is based on tight

[4] measured on an Intel Core2 Duo CPU T7700 2.4 GHz, 2.0 GB RAM, Windows XP Professional, SP3, Java 1.6; time for reading the input file, establishing the constraints, propagation and search.

envelopes of the inventory level of the reservoir. Runtime examinations on an accepted benchmark set [7] show the benefit of this algorithm. Current and future work focuses on an adaptation of the maximum flow approach for the level approximations used in the balance constraint as an alternative to the higher-order approximations proposed in [5]. First examinations show that the number of backtracking steps can be further reduced up to an order of magnitude.

References

1. Amedeo Cesta and Cristiano Stella. A time and resource problem for planning architectures. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, pages 117–129. Springer Verlag, 1997.
2. Rina Dechter, Italy Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, May 1991.
3. Matthias Hoche, Henry Müller, Hans Schlenker, and Armin Wolf. `firstCS` - A Pure Java Constraint Programming Engine. In Michael Hanus, Petra Hofstedt, and Armin Wolf, editors, *2nd International Workshop on Multi-paradigm Constraint Programming Languages – MultiCPL'03*. Online available at `uebb.cs.tu-berlin.de/MultiCPL03/Proceedings.MultiCPL03.RCoRP03.pdf`, 29th September 2003. (Last visited: 2009/05/13).
4. T. K. Satish Kumar. Incremental computation of resource-envelopes in producer-consumer models. In Francesca Rossi, editor, *Principles and Practice of Constaint Programming - CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 664–678. Springer Verlag, 2003.
5. Philippe Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artifical Intelligence*, 143:151–188, 2003.
6. Nicola Muscettola. Computing the envelope for stepwise-constant resource allocations. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 139–154. Springer-Verlag, 2002.
7. Klaus Neumann and Christoph Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56:513–533, 2002.
8. Helmut Simonis and Trijntje Cornelissens. Modelling producer/consumer constraints. In *Principles and Practice of Constraint Programming - CP'95, First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings*, volume 976 of *Lecture Notes in Computer Science*, pages 449–462. Springer, 1995.