Approximated Consistency for the Automatic Recording Problem

Meinolf Sellmann

Brown University, Department of Computer Science,
115 Waterman Street, P.O. Box 1910, Providence, RI 02912, U.S.A.
sello@cs.brown.edu

In constraint optimization, global constraints play a decisive role. To develop an efficient optimization tool, we need to be able to assess whether we are still able to improve the objective function further. This observation has lead to the development of a special kind of global constraints, so-called optimization constraints [2,5]. Roughly speaking, an optimization constraint expresses our wish to search for improving solutions only while enforcing feasibility for at least one of the constraints of the problem.

Since optimization constraints essentially evolve as a conjunction of a constraint on the objective value and some constraint of the constraint program, for many optimization constraints achieving generalized arc-consistency turns out to be NP-hard. Consequently, weaker notions of consistency have been developed with the aim to get ourselves back into the realm of tractable inference techniques. In [6,7], we introduced the concept of approximated consistency which is a refined and stronger notion of relaxed consistency [1] for optimization constraints. Approximated consistency asks that all assignments are removed from consideration whose commitment would cause a bound with guaranteed accuracy to drop below the given threshold.

We study the automatic recording problem (ARP) that consists in the solution of a knapsack problem where items are associated with time intervals and only items can be selected whose corresponding intervals do not overlap. The combination of a knapsack constraint with non-overlapping time-interval constraints can be identified as a subproblem in many more scheduling problems. For example, satellite scheduling can be viewed as a refinement of the automatic recording problem. Therefore, it is of general interest to study a global constraint that augments the knapsack constraint with time-interval consistency of selected items. This idea gives raise to the Automatic Recording Constraint (ARC), which we want to study in this paper. Obviously, as an augmentation of the knapsack constraint, achieving generalized arc-consistency for the ARC is NP-hard. Consequently, we will develop a filtering algorithm for the constraint that does not guarantee backtrack-free search for the ARP, but that achieves at least approximated consistency with respect to bounds of arbitrary accuracy.

1 ARP Approximation

In the interest of space, we need to omit formal definitions of optimization constraints and approximated consistency. We refer the reader to [1,7,8]. Let us define the Automatic Recording Problem and its corresponding constraint.

Given $n \in \mathbb{N}$, denote with $V = \{1, \ldots, n\}$ the set of *items*, and with $start(i) < end(i)\ \forall\ i \in V$ the corresponding starting and ending times. With $w = (w_i)_{1 \leq i \leq n} \in$

$\mathbb{N}_+^n$ we denote the storage requirements, $K \in \mathbb{N}_+$ denotes the storage capacity, and $p = (p_i)_{1 \le i \le n} \in \mathbb{N}^n$ the profit vector. Finally, let us define n binary variables $X_1, \ldots,$- $X_n \in \{0,1\}$. We say that the interval $I_i := [start(i), end(i)]$ *corresponds* to item $i \in V$, and call two items $i, j \in V$ *overlapping* whose corresponding intervals overlap, i.e. $I_i \cap I_j \neq \emptyset$. We call $p_X := \sum_{i \mid X_i = 1} p_i$ the *user satisfaction (with respect to X)*.

The *Automatic Recording Problem (ARP)* consists in finding an assignment $X = (X_1, \ldots, X_n) \in \{0,1\}^n$ such that (a) The selection X can be stored within the given disc size, i.e. $\sum_i w_i X_i \le K$. (b) At most one item must be selected at a time, i.e. $I_i \cap I_j = \emptyset \ \forall\, i < j$ s.t. $X_i = 1 = X_j$. (c) X maximizes the user satisfaction, i.e. $p_X \ge p_Y \ \forall\, Y$ respecting (a) and (b). Then, given a lower bound on the objective function $B \in \mathbb{N}$ and domains of the binary variables $X_1, \ldots, X_n$, the Automatic Recording Constraint (ARC) consists in enforcing that a solution to the ARP with $p_X > B$.

In less formal terms, the ARC requires us to find a selection of items such that the total weight limit is not exceeded, no two items overlap in time, and the total objective value is greater than that of the best know feasible solution. Note that enforcing generalized arc-consistency (GAC) on the ARC is NP-hard, which is easy to see by the fact that finding an improving solution would otherwise be possible in a backtrack-free search [3] or by simple reduction to the knapsack constraint [7].

Approximated consistency requires that a lower threshold that is diminished by some fraction of the overall best possible performance is guaranteed to be exceeded. In our pursuit to develop a filtering algorithm for the ARC, let us first study the ARP and see whether we can develop a fast approximation algorithm for the problem. Let $p_{max} := \max\{p_i \mid 1 \le i \le n\}$. We develop a pseudo-polynomial algorithm running in $\Theta(n^2 p_{max})$ that will be used later to derive a fully polynomial time approximation scheme (FPTAS) for the ARP.

1.1 A Dynamic Programming Algorithm

The algorithm we develop in the following is similar to the teaching book dynamic programming algorithm for knapsack problems. Setting $\overline{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$ and $\psi := np_{max} + 1$, we compute a matrix $M = (m_{kl}) \in \overline{\mathbb{N}}^{n+1 \times \psi}$, $0 \le k \le \psi$, $0 \le l \le n$. In m_{kl}, we store the minimal knapsack capacity that is needed to achieve a profit greater or equal k using items lower or equal l only ($m_{kl} = \infty$ iff $\sum_{1 \le i \le l} p_i < k$).

We assume that V is ordered with respect to increasing ending times, i.e., $1 \le i < j \le n$ implies $e_i \le e_j$. Further, denote with $last_j \in V \cup \{0\}$ the last non-overlapping node lower than j, i.e., $e_{last_j} < s_j$ and $e_i \ge s_j \ \forall\, last_j < i \le j$.

We set $last_j := 0$ iff no such node exists, i.e., iff $e_0 \ge s_j$. To simplify the notation, let us assume that $m_{k,0} = \infty$ for all $0 < k < \psi$, and $m_{k,0} = 0$ for all $k \le 0$. Then,

$$m_{kl} = \min\{m_{k,l-1}, m_{k-p_l, last_l} + w_l\}. \tag{1}$$

The above recursion equation yields a dynamic programming algorithm: First, we sort the items with respect to their ending times and determine $last_i$ for all $0 \le i < n$. Both can be done in time $\Theta(n \log n)$. Then, we build up the matrix column by column, and within each column from top to bottom. Finally, we compute $\max\{k \mid m_{k,n} \le K\}$. The total running time of this procedure and the memory needed are obviously in $\Theta(|M|) = \Theta(n^2 p_{max})$.

1.2 A Fully Polynomial Time Approximation Scheme

We exploit a core idea from [4] to limit the total number of non-infinity entries per column: According to Equation 1, each column depends solely on the column immediately to the left and the column that belongs to the last predecessor of the currently newly added item. We construct new sparse columns as lists of only non-infinity entries that are ordered with increasing profit. This can be done easily by running through the corresponding lists of columns that determine the entries in the new column. After a new column is created, we "trim" it by eliminating entries whose profit (the corresponding row of the matrix entry) is only slightly better than that of another entry in the column. Formally, we remove an entry if there exists a prior entry m_{kl} in the list if and only if there exists a prior and not previously removed entry m_{jl} such that $j \geq (1-\delta)k$ for some $1 > \delta = \delta(\varepsilon) > 0$. And whenever an entry m_{kl} is removed, its representant entry is set to $m_{jl} := \min\{m_{jl,m_{kl}}\}$. All this can be done in one linear top to bottom pass through the column. Then, after the trimming, successive elements in the list differ by a factor of at least $1/(1-\delta)$. Thus, each sparse column can contain at most $\log_{1/(1-\delta)}(np_{max}) = \frac{\ln(np_{max})}{-\ln(1-\delta)} \leq \frac{\ln(np_{max})}{\delta}$ elements. Note that every new column, before it is trimmed itself, cannot contain more than two times this value. Consequently, the algorithm will only take time $O(\frac{n\ln(np_{max})}{\delta})$.

Now, what error have we introduced by trimming the columns? By induction on the column indices l, it can shown easily that in the lth column, if there existed an entry m_{kl} in the original dynamic program, then there exists an entry m_{jl} in the trimmed version such that $(1-\delta)^l k \leq l \leq k$ and $m_{jl} \leq m_{kl}$. Consequently, the entry $m_{kn} \leq K$ that achieves the optimal profit k has a representant $m_{ln} \leq m_{kl} \leq K$ with $l \geq (1-\delta)^n k$. When setting $\delta = \varepsilon/n$, then it follows $l \geq (1-\frac{\varepsilon}{n})^n k \geq (1-\varepsilon)k$. Consequently, we achieve an FPTAS that runs in time $O(\frac{n^2\ln(np_{max})}{\varepsilon})$.

2 Approximated Consistency for the ARC

In order to achieve a filtering algorithm for the ARC based on the routine that we developed before, we closely follow the idea of defining a directed acyclic graph over the trimmed dynamic programming matrix. The idea was first introduced in [9] and consequently lead to the filtering algorithms in [7].

We define the weighted, directed, and acyclic graph for the untrimmed matrix as follows: Every non-infinity entry in the matrix defines a node in the graph. In accordance to Equation 1, each node has at most two incoming arcs: one emanating from the column immediately to the left, and another emanating from the column that corresponds to the last predecessor of the item that is newly added in the current column (whereby the column of the last predecessor may be identical to the column immediately to the left). That is, one incoming arc represents the decision not to use the newly added item (we refer to those arcs as zero-arcs), and the other incoming arc represents the decision to add the new item corresponding to the new column (we refer to those arcs as one-arcs). A zero-arc has associated weight 0, a one-arc has the same weight as the item corresponding to the column the target node belongs to. To express that we are only looking for solutions with profit greater B, we add a sink node t and connect it to the graph by directing arcs to t from exactly those nodes in the last column that have profit greater B. With this construction, we ensure a one-to-one correspondence between solutions to the ARP and paths in the graph: A feasible, improving solution corresponds exactly to a path from m_{00} to t that has weight lower or equal K. We call such paths *admissible*. The original numbers in the dynamic programming matrix now correspond to shortest-path distances from $m_{0,0}$ to the individual nodes.

The question that arises is how we can incorporate the idea of trimming a column. Trimming removes nodes from columns so as to make sure that the number of non-infinity entries stays polynomial in every column. We would like to trim, but we must make sure that by removing nodes we do not eliminate arcs from the graph that could actually belong to admissible paths. Otherwise we may end up filtering values from variable domains that could actually lead to improving, feasible solutions with respect to the ARC, i.e. our filtering algorithm could filter incorrectly, which we must prevent.

The algorithm that we propose uses the trimming idea as follows: In the graph, whenever a column entry would be removed by trimming, we keep the respective node and add an arc with weight 0 to the node that represents the node. The trimmed node has no other outgoing arcs, especially none that target outside of the column that it belongs to. This implies that, for new columns that are generated later, the trimmed node is of no relevance, so that we can still keep the column fill-in under control. With this slight modification of the graph, we can ensure that it has polynomial size and that the filtering method achieves ε-consistency for the ARC.

Let us formalize the idea by defining the graph corresponding to the trimmed dynamic program as follows. We define the weighted, directed, and acyclic graph $G(\delta) = (N, A, v)$ (whereby we always only consider nodes m_{qk} which have a non-infinity value in the dynamic program) by setting: $N_R := \{m_{qk} \mid 0 \leq q \leq np_{max}, 0 \leq k \leq n, m_{qk}\ \delta\text{−untrimmed}\}$. $N_T := \{m_{qk} \mid 0 \leq q \leq np_{max}, 0 \leq k \leq n, m_{qk}\ \delta\text{−trimmed}\}$. $N := N_R \cup N_T \cup \{t\}$. $A_0 := \{(m_{q,k-1}, m_{qk}) \mid k \geq 1,\ m_{q,k-1} \in N_R,\ m_{qk} \in N_R \cup N_T\}$. $A_1 := \{(m_{q-p_k,last_k}, m_{qk}) \mid k \geq 1,\ q \geq p_k,\ m_{q-p_k,last_k} \in N_R,\ m_{qk} \in N_R \cup N_T\}$. $A_R := \{(m_{pk}, m_{qk}) \mid m_{pk} \in N_T,\ m_{qk} \in N_R,\ (1-\delta)p \leq q < p\}$. $A_t := \{(m_{qn}, t) \mid q \geq B,\ m_{qn} \in N_R\}$. $A := A_0 \cup A_1 \cup A_R \cup A_t$. $v(e) := 0$ for all $e \in A_0 \cup A_R \cup A_t$. $v(m_{q-p_k,last_k}, m_{qk}) := w_k$ for all $(m_{q-p_k,last_k}, m_{qk}) \in A_1$.

Note that, for an admissible path $(m_{00}, \ldots, m_{pn}, t)$ in the graph, the sequence of arcs in A_0 and A_1 (whereby we ignore arcs in A_R and A_t) determines the *corresponding solution* to the ARP when we set all items that belong to skipped columns to 0. That corresponding solution then has the same weight as the path, and according to Section 1.2 for the corresponding solutions profit q it holds: $(1-\varepsilon)q \leq p \leq q$.

Theorem 1. *Given* $1 > \varepsilon > 0$*, we set* $\delta := \varepsilon/n$*.*

1. *If there exists a path* $W = (m_{00}, \ldots, m_{pn}, t)$ *in* $G(0)$ *with* $p \geq B$ *and such that* $v(W) \leq K$*, then there exists a path* $X = (m_{00}, \ldots, m_{qn}, t)$ *in* $G(\delta)$ *such that* $q \geq (1-\varepsilon)B$*,* $v(X) \leq K$*, and the corresponding solutions to* W *and* X *are identical.*
2. *If there exists a path* $X = (m_{00}, \ldots, m_{qn}, t)$ *in* $G(\delta)$ *with* $q \geq (1-\varepsilon)B$ *and such that* $v(X) \leq K$*, then there exists a path* $W = (m_{00}, \ldots, m_{pn}, t)$ *in* $G(0)$ *such that* $p \geq (1-\varepsilon)B$*,* $v(W) \leq K$*, and the corresponding solutions to* W *and* X *are identical.*

We exploit this theorem to devise the following filtering algorithm: Thanks to the fact that our graph is directed and acyclic, we can apply linear time shorter path filtering techniques that remove those and only those arcs that cannot be visited by any admissible path. After shorter path filtering, every arc in the pruned graph can be part of a path from m_{00} to t with weight lower or equal K. Since arcs really correspond to decisions to include or exclude and item in our solution, there exists a one-arc (zero-arc) to a node in column i iff item i is included (excluded) in some improving, feasible solution. Consequently, by searching the pruned graph for columns in which no node has incoming one-arcs, we can identify those and only those items that must be excluded in all improving, feasible solutions. The situation is only slightly more complicated when a column has no incoming zero-arcs. In contrast to knapsack approximation, for the ARC there exist arcs that cross several columns. If there still exists such an arc that can be part of an admissible path, then the items that belong to the columns that are bridged can obviously be excluded in some admissible solution. Consequently, if a column has only incoming one-arcs and no arc crosses the column, then and only then it must indeed be included in all feasible improving solutions. Without going into details, we just note that the detection of items that must be included can be performed in time $O(n \log n + |M|)$.

According to Theorem 1 (1), this is a correct filtering algorithm for the ARC, and according to Theorem 1 (2) we are sure to eliminate all assignments that would cause the best optimal solution to drop below $(1-\varepsilon)B$. Assuming that B is given as a lower bound on the objective, i.e. $B \leq P^*$, we finally have:

Corollary 1. *Approximated consistency for the ARC can be achieved in time* $O(n^2 \ln(np_{max})/\varepsilon)$.

References

1. T. Fahle and M. Sellmann. Cost-Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115:73–93, 2002.
2. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming (CP)* Springer LNCS 1713:189–203, 1999.
3. E.C. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29(1):24–32, 1982.
4. O.H. Ibarra and C.E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of the ACM*, 22(4):463–468, 1975.
5. M. Milano. *Integration of Mathematical Programming and Constraint Programming for Combinatorial Optimization Problems*, Tutorial at CP2000, 2000.
6. M. Sellmann. The Practice of Approximated Consistency for Knapsack Constraints. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, AAAI Press, pp. 179-184, 2004.
7. M. Sellmann. Approximated Consistency for Knapsack Constraints. *CP*, Springer LNCS 2833: 679–693, 2003.
8. M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for the Automatic Recording Problem. *Annals of Operations Research*, 118:17-33, 2003.
9. M. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *3rd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR)*, pp. 113–124, 2001.