

The Practice of Approximated Consistency for Knapsack Constraints *

Meinolf Sellmann

Cornell University, Department of Computer Science
4130 Upson Hall, Ithaca, NY 14853, U.S.A.
sello@cs.cornell.edu

Abstract

Knapsack constraints are a key modeling structure in discrete optimization and form the core of many real-life problem formulations. Only recently, a cost-based filtering algorithm for Knapsack constraints was published that is based on some previously developed approximation algorithms for the Knapsack problem. In this paper, we provide an empirical evaluation of approximated consistency for Knapsack constraints by applying it to the Market Split Problem and the Automatic Recording Problem.

Introduction

Knapsack constraints are a key modeling structure in discrete optimization and form the core of many real-life problem formulations. Especially in integer programming, a large number of models can be viewed as a conjunction of Knapsack constraints only. However, despite its huge practical importance, it has been given rather little attention by the Artificial Intelligence community, especially when comparing it to the vast amount of research that was conducted in the Operations Research and the Algorithms communities.

One of the few attempts that were made was published in (Sellmann 2003). In this paper, we introduced the theoretical concept of *approximated consistency*. The core idea of this contribution consists in using bounds of guaranteed accuracy for cost-based filtering of optimization constraints. As the first example, the paper introduced a cost-based filtering algorithm for Knapsack constraints that is based on some previously developed approximation algorithms for the Knapsack problem.

It was shown theoretically that our algorithm achieves approximated consistency in amortized linear time for bounds with arbitrary but constant accuracy. However, no practical evaluation was provided, and until today it is an open question whether approximated consistency is just a beautiful theoretical concept, or whether it can actually yield performant cost-based filtering algorithms.

In this paper, we provide an empirical evaluation of approximated consistency for Knapsack constraints. After

briefly reviewing the filtering algorithm presented in (Sellmann 2003), we discuss practical enhancements of the filtering algorithm. We then use our implementation and apply it to two very different problems: First, we devise an algorithm for the Market Split Problem (MSP) and compare it with previous algorithms for the problem based on mathematical programming or constraint programming. Since the MSP represents a computationally very hard problem for which already very small problem instances cause severe problems for standard solution techniques, next we apply approximated Knapsack filtering to the much simpler Automatic Recording Problem (ARP). We compare the performance of Knapsack filtering algorithms based on approximated consistency with filtering algorithms based on linear relaxation bounds.

A Brief Review of Approximated Consistency for Knapsack Constraints

We start by reviewing approximated consistency for Knapsack constraints as it was introduced in (Sellmann 2003). In the following, denote with $n \in \mathbb{N}$ the number of items in our Knapsack, denote with $w^T = (w_1, \dots, w_n) \in \mathbb{N}^n$ the weights and with $p^T = (p_1, \dots, p_n) \in \mathbb{N}^n$ the profits of the items. Furthermore, denote with C the capacity of the Knapsack, and denote with B a lower bound on the objective.

Definition 1

Given n binary variables $X^T = (X_1, \dots, X_n)$, we define: A Knapsack Constraint $\mathcal{KC}(X_1, \dots, X_n, w_1, \dots, w_n, C, p_1, \dots, p_n, B)$ is true, iff $w^T X \leq C$ and $p^T X \geq B$.

A Knapsack constraint (KC) is the formal expression of our wish to search for a feasible (wrt. to the limited capacity), improving solution. Therefore, we also speak of the KC as an *optimization constraint* (Focacci *et al.* 1999). Now, a prominent task in Constraint Programming is to identify all variable assignments that would yield to an inconsistency. With respect to the KC, these are all decisions to include or remove an item that would either cause the capacity of the knapsack to be exceeded, or that, for all solutions that are feasible with respect to the limited capacity, would force the objective function to drop below B . For optimization constraints, this process of eliminating non-profitable variable assignments is called *cost-based filtering*.

*This work was supported by the Intelligent Information Systems Institute (AFOSR Grant F49620-01-1-0076).
Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Clearly, cost-based filtering for KCs is an NP-hard task, since it requires the solution of some Knapsack problems itself. Therefore, we suggested to relax the requirement that all non-profitable assignments be found and introduced the notion of *approximated consistency*:

Definition 2

Denote with $\mathcal{KC}(X_1, \dots, X_n, w_1, \dots, w_n, C, p_1, \dots, p_n, B)$ a KC where the variables X_i are currently allowed to take values in $D_i \subseteq \{0, 1\}$. Then, given some $\epsilon \geq 0$, we say that \mathcal{KC} is ϵ -consistent, iff for all $1 \leq i \leq n$ and $x_i \in D_i$ there exist $x_j \in D_j$ for all $j \neq i$ such that

$$\sum_{i \leq n} w_i x_i \leq C \quad \text{and} \quad \sum_{i \leq n} p_i x_i \geq B - \epsilon P^*,$$

whereby $P^* = \max\{\sum p_i y_i \mid y_i \in D_i, \sum w_i y_i \leq C\}$.

Therefore, to achieve a state of ϵ -consistency for a KC, we must ensure that

1. all items are deleted that cannot be part of any solution that obeys the capacity constraint and that achieves a profit of at least $B - \epsilon P^*$, and
2. all items have to be permanently inserted into the knapsack that are included in all solutions that obey the capacity constraint and that have a profit of at least $B - \epsilon P^*$.

That is, we do not enforce that all variable assignments are filtered that do not yield to any improving solution, but at least we want to remove all assignments for which the performance is forced to drop too far below the critical objective value. It is this property of filtering against a bound of guaranteed accuracy (controlled by the parameter ϵ) that distinguishes the filtering algorithms in (Sellmann 2003) from earlier cost-based filtering algorithms for KCs that were based on linear programming bounds (Fahle and Sellmann 2002).

Of course, it would be nicer if we could use the stronger bound $(1 - \epsilon)B$, but for small lower bound values B we do not know how to perform filtering based on this bound efficiently. Therefore, the optimal Knapsack value P^* is used in the definition of approximated consistency. In the filtering algorithm, we cannot afford to compute P^* of course (since this would require to solve the Knapsack problem), but we can nevertheless prove that this bound is achieved even without computing it exactly.

Now, to achieve a state of ϵ -consistency for a KC, we modified an approximation algorithm for Knapsacks developed in (Lawler 1977). This algorithm works in two steps: First, the items are partitioned into two sets of large (L) and small (S) items, whereby L contains all items with profit larger than some threshold value T . The profits of the items in L are scaled and rounded down by some factor K : $\bar{p}_L := \lfloor p_L / K \rfloor$.

In the second step, the algorithm computes a solution $x = (x_L, x_S) \in \{0, 1\}^n$ (whereby v_L and v_S contain all entries of a vector v whose corresponding indices are in L or S , respectively) such that:

- If an item in S is included in the solution then so are all items in S that have a higher efficiency (whereby the efficiency of an item is its profit divided by its weight), and

- $K\bar{p}_L^T x_L + p_S^T x_S$ is maximized.

If T and K are chosen carefully, the scaling of the profits of items in L and the additional constraint that a solution can only include items in S with highest efficiency make it possible to solve this problem in polynomial time. Moreover, it can be shown that the solution obtained has an objective value within a factor of $(1 - \epsilon)$ from the optimal solution of the Knapsack that we denoted with P^* (Lawler 1977).

The cost-based filtering algorithm makes use of the performance guarantee given by the algorithm. Clearly, if the solution quality of the approximation under some variable assignment drops below $B - \epsilon P^*$, then this assignment cannot be completed to an improving, feasible solution, and it can be eliminated. We refer the reader to (Sellmann 2003) for further details on how this elimination process can be performed efficiently.

Practical Enhancements of the Filtering Algorithm

The observation that the filtering algorithm is based on the bound provided by the approximation algorithm is crucial. It means that, in contrast to the approximation algorithm itself, we hope to find rather bad solutions by the approximation, since they provide more accurate bounds on the objective function. The important task when implementing the filtering algorithm is to estimate the accuracy achieved by the approximated solution as precisely as possible. For this purpose, in the following we investigate three different aspects:

- the computation of a 2-approximation that is used to determine K and the filtering bound,
- the choice of the scaling factor K , and
- the computation of the filtering bound.

Computation of a 2-Approximation

A 2-approximation on the given Knapsack is used in the approximation algorithm for several purposes. Most importantly, it is used to set the actual filtering bound \bar{B} . From our discussion in the previous section it is clear that \bar{B} can be set to $B - \epsilon P^*$. However, we cannot afford to compute P^* since this would require to solve the Knapsack problem first. Therefore, we showed that it is still correct to filter an assignment if the performance drops below $B - \epsilon P_0$ only, whereby P_0 denotes the value of a 2-approximation of the original Knapsack problem.

The standard procedure for computing a 2-approximation P_0 is to include items with respect to decreasing efficiency. If an item fits, we insert it, otherwise we leave it out and consider the following items. After all items have been considered, we compare the solution obtained in that way with the Knapsack that is obtained when we follow the same procedure but by now considering the items with respect to decreasing profit (whereby we assume that $\|w\|_\infty \leq C^1$). Then, we choose P_0 as the solution that has the larger objective function value. It is easy to prove that this procedure yields a 2-approximation.

¹ $\|w\|_\infty$ denotes the maximum absolute entry of vector w .

Now, with respect to the filtering routine, of course we would like to set P_0 as small as possible. Note that it is of no relevance for the correctness of the filtering algorithm that there actually exists a solution that has value P_0 . All that is required is that $2P_0$ is an upper bound which is indeed essential for the correctness of our algorithm. Therefore, we propose to compute the value P_0 in a much simpler way. We can obtain an upper bound by allowing that items can be inserted fractionally and solving the relaxed problem as a linear problem. We obtain an upper bound U , and we simply set $P_0 := U/2$. It is easy to show that this procedure returns a value for P_0 that is at most as large as the value returned by the standard 2-approximation algorithm.

Setting the Scaling Factor

The next issue regards the factor with which the profits in the set of large items L are scaled down. According to the correctness proof in (Sellmann 2003), we need to ensure that $\frac{K}{T} p_L^T x_L^* + p_S^T (x_S^* - \bar{x}_S) \leq \epsilon P_0$, whereby x^* denotes an optimal solution to the Knapsack problem, and \bar{x} denotes the solution found by our approximation algorithm. We proposed to set $K := \frac{\epsilon}{4} T$. Intuitively, we obtain worse approximation results (and consequently more accurate upper bounds) and shorter running times if we scale the profits of the items in L as much as possible. We can choose the scaling factor more aggressively, by setting $K := \frac{\epsilon P_0}{2U_L} T$, where U_L denotes an upper bound on the large item Knapsack problem. Then, it holds that:

$$\frac{K}{T} p_L^T x_L^* + p_S^T (x_S^* - \bar{x}_S) \leq \frac{\epsilon P_0}{2} \frac{p_L^T x_L^*}{U_L} + \frac{\epsilon P_0}{2} \leq \epsilon P_0.$$

Therefore, our setting maintains correctness. In order to compute U_L we use the linear programming upper bound on the large item problem. Nota bene: If $U_L = 0$, the large item problem is trivially solved by the empty Knapsack, and we do not need to scale the profits at all.

Setting the Filtering Bound

The last important issue regards the bound against which we filter eventually. In the analysis of the approximation algorithm both the large and the small item problem contribute equally to the approximation error made. More precisely, the large item problem contributes an error of $\frac{\epsilon P_0}{2}$, and the small item problem contributes $\|p_S\|_\infty$. When setting $T := \frac{\epsilon P_0}{2}$, clearly the total deviation from the optimum is bounded by ϵP_0 . Consequently, we proposed to set the filtering bound to $\bar{B} := B - \epsilon P_0$. With respect to the brief discussion above, we can already enhance on this by setting $\bar{B} := B - \frac{\epsilon P_0}{2} - \|p_S\|_\infty$. Like that, if the small item problem is actually empty, we are able to half the absolute error made! Similarly, if the large item problem is empty, or if $U_L = 0$, this problem does not contribute to the error made. Then we can even set $\bar{B} := B - \|p_S\|_\infty$.

Approximated Knapsack Filtering for Market Split Problems

We implemented the approximated filtering algorithm including the enhancements discussed in the previous sec-

tion. To evaluate the practical usefulness of the algorithm, we first apply it in the context of Market Split Problems (MSPs), a benchmark that was suggested for Knapsack constraints in (Trick 2001). The original definition goes back to (Cornuéjols and Dawande 1998; Williams 1978): A large company has two divisions D_1 and D_2 . The company supplies retailers with several products. The goal is to allocate each retailer to either division D_1 or D_2 so that D_1 controls $A\%$ of the company's market for each product and D_2 the remaining $(100-A)\%$. Formulated as an integer program, the problem reads:

$$\begin{aligned} \sum_j a_{ij} x_j &= \frac{A}{100} \sum_j a_{ij} & \forall 0 \leq i < m \\ x_j &\in \{0, 1\} & \forall 0 \leq j < n, \end{aligned}$$

whereby m denotes the number of products, n is the number of retailers, and a_{ij} is the demand of retailer j of product i .

A Tree-Search-Algorithm for the MSP

We model the problem as a constraint program by introducing one Knapsack constraint for each product, whereby profit and weight of each Knapsack item are equal. Capacity and profit bound are determined by the right hand side of the constraint. Following an idea presented in (Trick 2001), additionally we add one more redundant Knapsack constraint that is the weighted sum of the other constraints:

$$\begin{aligned} \sum_i \lambda^i \sum_j a_{ij} x_j &= \sum_i \lambda^i \frac{A}{100} \sum_j a_{ij} \\ x_j &\in \{0, 1\} & \forall 0 \leq j < n. \end{aligned}$$

In (Trick 2001) it was conjectured that, if λ was chosen large enough, the summed-up constraint contained the same set of solutions as the conjunction of Knapsack constraints. This is true for Knapsack constraints as they evolve in the context of the MSP where the weight and the profit of each item are the same and capacity and profit bound are matching. However, the example $3 \leq 2x_1 + 7x_2 \leq 8$ in combination with $3 \leq 7x_1 + 2x_2 \leq 8$, $x_1, x_2 \in \{0, 1\}$, shows that this is not true in general: The conjunction of both Knapsack constraints is clearly infeasible, yet no matter how we set λ , no summed-up constraint alone is able to reveal this fact. Note that, when setting λ to a very large value in the summed-up constraint, the profit becomes very large, too. Consequently, the absolute error made by the approximation algorithm becomes larger, too, and thereby renders the filtering algorithm less effective. In accordance to Trick's report, we found that $\lambda = 5$ yields very good filtering results for the Cornuéjols-Dawande MSP instances that we consider in our experimentation.

Using the model as discussed above, our algorithm for solving the MSP carries out a tree-search. In every choice point, the Knapsack constraints perform cost-based filtering, exchanging information about removed and included items. On top of that, we perform cost-based filtering by changing the capacity constraints within the Knapsack constraints. As Trick had noted in his paper already, the weights of the items and the capacity of the Knapsack can easily be modified once a Knapsack constraint is initialized. The new weight constraints considered are the capacity and profit constraints of the other Knapsack constraints as well as additional weighted sums of the original capacity constraints,

ϵ		# Constraints / # Variables – % Feasible					
		3 / 20 – 8%		4 / 30 – 29%		5 / 40 – 34%	
		CPs	Time	CPs	Time	CPs	Time
5%	max	11	0.18	8699	211.8	-	-
	ϕ	3.53	0.08	1392	50.21	-	-
	min	1	0.04	7	1.04	-	-
2%	max	3	0.07	321	31.53	505K	52.6K
	ϕ	1.29	0.03	26.71	4.92	30.1K	4867
	min	1	0.01	3	0.98	96	36.2
1%	max	3	0.05	19	7.68	26.8K	12.6K
	ϕ	1.25	0.03	2.7	0.68	943.3	552.8
	min	1	0.01	1	0.41	25	38.85
5‰	max	3	0.05	7	0.99	171	826.8
	ϕ	1.25	0.03	2.61	0.61	60.15	174.9
	min	1	0.02	1	0.42	5	30.96
2‰	max	3	0.04	7	0.96	59	47.19
	ϕ	1.25	0.03	2.61	0.61	13.9	28.99
	min	1	0.01	1	0.43	3	16.08
1‰	max	3	0.05	7	0.95	59	47.75
	ϕ	1.25	0.03	2.61	0.61	13.9	28.98
	min	1	0.01	1	0.34	3	16.19
0.5‰	max	-	-	7	0.95	59	47.8
	ϕ	-	-	2.61	0.61	13.9	28.98
	min	-	-	1	0.44	3	16.13

Table 1: Numerical results for our 3 benchmark sets. Each set contains 100 Cornuéjols-Dawande instances, and the percentage of feasible solutions in each set is given. We apply our MSP algorithm varying the approximation guarantee ϵ for the Knapsack filtering algorithm between 5% and 0.5‰. For each benchmark set, we report the maximum, average, and minimum number of choice points and computation time in seconds.

whereby we use as weights powers of $\lambda = 5$ again while considering the original constraints in random order. After the cost-based filtering phase, we branch by choosing a random item that is still undecided yet and continue our search in a depth-first manner.

Numerical Results for the MSP

All experiments in this paper were conducted on an AMD Athlon™ 1.2 GHz Processor with 500 MByte RAM running Linux 2.4.10, and we used the gnu C++ compiler version g++ 2.91.

(Cornuéjols and Dawande 1998) generated computationally very hard random MSP instances by setting $n := 10(m - 1)$, choosing the a_{ij} randomly from the interval $\{0, \dots, 99\}$, and setting $A := 50\%$. We use their method to generate 3 benchmark sets containing 3 products (constraints) and 20 retailers (variables), 4 constraints and 30 variables, and 5 constraints and 40 variables. Each set contains 100 random instances. In Table 1 we report our numerical results.

As our results show, the choice of ϵ has a severe impact on the performance of our algorithm. We see that for the benchmark set (3,20) it is completely sufficient to set $\epsilon := 0.02$. However, even for these very small problem instances, setting ϵ to 5% almost triples the average number of choice points and computation time. On the other hand, choosing a better approximation guarantee does not improve on the

performance of the algorithm anymore. We get a similar picture when looking at the other benchmark sets, but shifted to a different scale: For the set (4,30), the best performance can be achieved when setting $\epsilon := 5\%$, for the set (5,40) this value even decreases to 2‰.

In our experiments, we found that choosing higher approximation accuracies can sometimes yield an increase in the number of choice points. This counter intuitive effect is explained by the fact that the approximated filtering algorithm is not monotone in the sense that demanding higher accuracy does not automatically result in better bounds. It may happen occasionally that for a demanded accuracy of 5‰ the approximation algorithm does a rather poor job which results in very tight bound on the objective function. Now, when demanding an accuracy of 1‰, the approximation algorithm may actually compute the optimal solution, thus yielding a bound that can be up to 1‰ away from the correct value. Consequently, by demanding higher accuracy we may actually get a lower one in practice.

When comparing the absolute running times with other results on the Cornuéjols-Dawande instances reported in the literature, we find that approximated consistency is doing a remarkably good job: Comparing the results with those reported in (Trick 2001), we find that approximated Knapsack filtering even outperforms the generate-and-test method on the (4,30) benchmark. Note that this method is probably not suitable for tackling larger problem instances, while approximated consistency allows us to scale this limit up to 5 constraints and 40 variables.

When comparing our MSP algorithm with the best algorithm that exists for this problem (Aardal *et al.* 1999), we find that, for the (4,30) and the (5,40) benchmark sets, we achieve competitive running times. This is a very remarkable result, since really approximated Knapsack filtering is in no way limited or special to Market Split Problems. It could for example also be used to tackle slightly different and more realistic problems in which a splitting range is specified rather than an exact percentage on how the market is to be partitioned. And of course, it can also be used when the profit of the items does not match their weight and in any context where Knapsack constraints occur.

We also tried to apply our algorithm to another benchmark set with 6 constraints and 50 variables. For this set, we were not able to experiment with values for ϵ that are below 5‰ since then the memory requirements of the filtering algorithm were exceeding the available RAM. We therefore note that the comparably high memory requirements present a clear limitation of the practical use of the approximated filtering routine. Still, the (6,50)-instances can be solved with our approach, but this takes a few hours on average. That this is possible at all is a direct merit of the improvements that we suggested in the previous section. Unfortunately, there is not space enough to present a detailed evaluation here. So we just note our improvements reduce the memory requirements and the accuracy of the algorithm considerably.

It remains to note that our experimentation also confirms a result from (Aardal *et al.* 1999) regarding the probability of

generating infeasible solutions by the Cornuéjols-Dawande method. Clearly, the number of feasible instances increases the more constraints and variables are considered by their generator. For a detailed study on where the actual phase-transition for Market Split Problems is probably located, we refer the reader to the Aardal paper.

Approximated Knapsack Filtering for the Automatic Recording Problem

Encouraged by the surprisingly good numerical results on the MSP, we want to evaluate the use of approximated consistency in the context of a very different application problem that incorporates a Knapsack constraint in combination with other constraints. We consider the Automatic Recording Problem (ARP) that was introduced in (Sellmann and Fahle 2003):

The technology of digital television offers to hide metadata in the content stream. For example, an electronic program guide with broadcasting times and program annotation can be transmitted. An intelligent video recorder like the TIVOtm system (TIVO) can exploit this information and automatically record TV content that matches the profile of the system's user. Given a profit value for each program within a predefined planning horizon, the system has to make the choice which programs shall be recorded, whereby two restrictions have to be met:

- The disk capacity of the recorder must not be exceeded.
- Only one program can be recorded at a time.

CP-based Lagrangian Relaxation for the ARP

Like in (Sellmann and Fahle 2003), we use an approach featuring CP-based Lagrangian relaxation: The algorithm performs a tree search, filtering out assignments, in every choice point, with respect to a Knapsack constraint (capturing the limited disk space requirement) and a weighted stable set constraint on an interval graph (that ensures that only temporally non-overlapping programs are recorded).

For the experiments, we compare two variants of this algorithm: The first uses approximated Knapsack filtering, the second a cost-based filtering method for Knapsack constraints that is based on linear relaxation bounds (Fahle and Sellmann 2002). While CP-based Lagrangian relaxation leaves the weight constraint unmodified but calls to the filtering routine with ever changing profit constraints, we replace, in the Knapsack constraint, each variable x_i by $y_i = 1 - x_i$. Thereby, profit and weight constraint change roles and we can apply the technique in (Trick 2001) again by calling the approximated Knapsack filtering routine with different capacity constraints.

Numerical Results for the ARP

For our experiments, we use a benchmark set that we made accessible to the research community in (ARP 2004). This benchmark was generated using the same generator that was used for the experimentation in (Sellmann and Fahle 2003):

Each set of instances is generated by specifying the time horizon (half a day or a full day) and the number of channels (20 or 50). The generator sequentially fills the channels

by starting each new program one minute after the last. For each new program, a *class* is being chosen randomly. That class then determines the interval from which the length is chosen randomly. The generator considers 5 different classes. The lengths of programs in the classes vary from 5 ± 2 minutes to 150 ± 50 minutes. The disk space necessary to store each program equals its length, and the storage capacity is randomly chosen as 45%–55% of the entire time horizon.

To achieve a complete instance, it remains to choose the associated profits of programs. Four different strategies for the computation of an objective function are available:

- For the *class usefulness (CU)* instances, the associated profit values are determined with respect to the chosen class, where the associated profit values of a class can vary between zero and 600 ± 200 .
- In the *time strongly correlated (TSC)* instances, each 15 minute time interval is assigned a random value between 0 and 10. Then the profit of a program is determined as the sum of all intervals that program has a non-empty intersection with.
- For the *time weakly correlated (TWC)* instances, that value is perturbed by a noise of $\pm 20\%$.
- Finally, in the *strongly correlated (SC)* data, the profit of a program simply equals its length.

Table 2 shows our numerical results. We apply our ARP algorithm using approximated consistency for cost-based filtering and vary ϵ between 5% and 5%. We compare the number of choice points and the time needed by this method with a variant of our ARP algorithm that uses Knapsack filtering based on linear relaxation bounds instead of approximated consistency. The numbers reported are the average and (in brackets) the median relative performance for each set containing 10 randomly generated instances.

Consider as an example the set containing instances with 50 channels and a planning horizon of 1440 minutes where the profit values have been generated according to method TWC. According to Table 2, the variant of our algorithm using approximated consistency with accuracy $\epsilon = 5\%$ visits 72.79% of the number of choice points that the variant incorporating linear programming bounds is exploring. However, when comparing the time needed, we see that approximated consistency actually takes 178.80% of the time, clearly because the time per choice point is higher compared to the linear programming variant.

Looking at the table as a whole, this observation holds for the entire benchmark set: even when approximated consistency is able to reduce the number of choice points significantly (as it is the case for the TWC and TSC instances), the effort is never (!) worthwhile. On the contrary, the approximation variant is, on average, 30% up to almost 5 times slower than the algorithm using linear bounds for filtering. In this context it is also remarkable that, in general, a variation of the approximation guarantee had no significant effect on the number of choice points visited. This effect is caused by the fact that an improvement on the Knapsack side need not necessarily improve the global Lagrangian bound.

obj.	#channels/ horizon	ϕ # progs	Time (sec)	$\epsilon = 5\%$		$\epsilon = 1\%$		$\epsilon = 5\%$	
				CPs (%)	Time (%)	CPs (%)	Time (%)	CPs (%)	Time (%)
CU	20/720	305.3	14.3	99.7 (104.6)	163.6 (153.6)	99.7 (104.6)	235.0 (224.6)	99.7 (104.6)	291.9 (280.0)
	50/720	771.3	58.7	101.5 (100.3)	168.5 (169.0)	101.5 (100.3)	169.6 (170.9)	101.5 (100.3)	227.9 (217.4)
	20/1440	609.8	67.8	97.2 (99.0)	193.3 (195.5)	97.2 (99.0)	234.5 (235.3)	97.2 (99.0)	359.7 (358.9)
	50/1440	1440	2426	118.1 (110.3)	165.2 (143.0)	118.1 (110.3)	165.2 (140.9)	112.2 (105.4)	165.9 (134.7)
SC	20/720	317.5	4.3	138.8 (122.5)	199.2 (125.8)	138.8 (122.5)	157.1 (541.9)	138.8 (122.5)	236.4 (1093.7)
	50/720	788.5	2.1	137.3 (128.9)	250.1 (242.3)	137.3 (128.9)	251.2 (245.0)	137.3 (128.9)	171.9 (732.9)
	20/1440	609.9	4.1	124.8 (114.2)	256.6 (317.5)	124.8 (114.2)	132.7 (595.2)	124.8 (114.2)	324.3 (1683.6)
	50/1440	1464	1.8	108.9 (108.3)	278.2 (365.7)	108.9 (108.3)	280.9 (371.3)	108.9 (108.3)	302.7 (405.2)
TWC	20/720	305.3	15.2	92.6 (94.1)	212.9 (249.5)	92.5 (94.1)	314.3 (340.6)	92.0 (94.1)	420.7 (388.0)
	50/720	771.3	60.8	72.8 (76.7)	178.8 (181.1)	72.8 (76.7)	181.5 (184.3)	72.8 (76.7)	298.1 (288.6)
	20/1440	609.8	207.0	83.4 (83.7)	189.4 (189.5)	83.4 (83.7)	325.4 (297.7)	83.4 (83.7)	217.4 (538.8)
	50/1440	1440	71.2	77.5 (80.2)	177.3 (181.4)	77.5 (80.2)	177.9 (181.9)	77.5 (80.2)	205.0 (206.8)
TSC	20/720	307.2	35.5	71.2 (71.4)	199.3 (182.6)	71.2 (71.4)	286.4 (268.9)	69.4 (71.4)	385.5 (370.3)
	50/720	780.8	681	84.7 (87.3)	144.5 (133.1)	84.7 (87.3)	141.5 (131.7)	84.7 (87.3)	192.0 (182.5)
	20/1440	609.8	239.5	82.6 (85.2)	172.0 (171.5)	82.6 (85.2)	267.0 (248.8)	82.6 (85.2)	480.3 (467.6)
	50/1440	1448	7295	91.9 (98.4)	134.6 (128.0)	91.9 (98.4)	134.7 (128.0)	91.9 (98.4)	140.6 (134.5)

Table 2: Numerical results for the ARP benchmark. We apply our algorithm varying the approximation guarantee of the Knapsack filtering algorithm between 5% and 5%. The first two columns specify the settings of the random benchmark generator: the first value denotes the objective variant taken, and it is followed by the number of TV channels and the planning time horizon in minutes. Each set identified by these parameters contains 10 instances, the average number of programs per set is given in the third column. We report the average (median) factor (in %) that the approximation variant needs relative to the variant using Knapsack filtering based on linear relaxation bounds. The average absolute time (in seconds) of the last algorithm is given in the fourth column.

It clearly depends on the application when a more accurate filtering of the Knapsack constraint pays off for the overall problem.

Conclusions

The most important advantage of approximated consistency is our ability to trade filtering efficiency for computation time. It would be most interesting to have a general understanding of this trade-off that would allow us to develop automated strategies for tuning the filtering effectiveness towards problems and problem instances.

Our numerical evaluation shows that manually tuned approximated consistency for Knapsack constraints can lead to massive improvements for critically constrained problems. Approximated consistency is the first generic constraint programming approach that can efficiently tackle the Market Split Problem (MSP), and is even competitive when compared with specialized state-of-the-art approaches for this problem. Note that, for the Knapsack constraints as they evolve from the MSP where profit and weight of each item are the same, linear relaxation bounds are in most cases meaningless, thus rendering Knapsack filtering based on these bounds without effect.

On the other hand, the application to the Automatic Recording Problem showed that, for combined problems, the effectiveness of a more accurate Knapsack filtering can be limited by the strength of the global bound that is achieved. The practical use of approximated filtering is further limited by high memory requirements for very tight approximation guarantees. And for lower accuracies, Knapsack filtering based on linear relaxation bounds may be almost as effective and overall faster.

References

- K. Aardal, R.E. Bixby, C.A.J. Hurkens, A.K. Lenstra, J.W. Smeltink. Market Split and Basis Reduction: Towards a Solution of the Cornu ejols-Dawande Instances. *IPCO*, Springer LNCS 1610:1–16, 1999.
- ARP: A Benchmark Set for the Automatic Recording Problem, maintained by M. Sellmann, http://www.cs.cornell.edu/~sello/-arp_benchmark.tar.gz
- G. Cornu ejols, M. Dawande. A Class of Hard Small 0-1 Programs. *IPCO*, Springer LNCS 1412:284–293, 1998.
- T. Fahle, M. Sellmann. Cost-Based Filtering for the Constrained Knapsack Problem. *AOR*, 115:73–93, 2002.
- F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *CP*, Springer LNCS 1713:189–203, 1999.
- E.L. Lawler. Fast Approximation Algorithm for Knapsack Problems. *FOCS*, pp. 206–213, 1977.
- M. Sellmann. Approximated Consistency for Knapsack Constraints. *CP*, Springer LNCS 2833: 679–693, 2003.
- M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for the Automatic Recording Problem. *AOR*, 118:17–33, 2003.
- TIVO. TV your way. TIVO, Inc., <http://www.tivo.com/home.asp>.
- M. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *CP-AI-OR’01*, pp. 113–124, 2001.
- H.P. Williams. Model Building in Mathematical Programming. Wiley, 1978.