Alcom-FT Technical Report Series
ALCOMFT-TR-03-174

Constraint Programming based Lagrangian Relaxation for the Automatic Recording Problem*

Meinolf Sellmann and Torsten Fahle

University of Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn
{sello,tef}@uni-paderborn.de

Abstract. Whereas CP methods are strong with respect to the detection of local infeasibilities, OR approaches have powerful optimization abilities that ground on tight global bounds on the objective. An integration of propagation ideas from CP and Lagrangian relaxation techniques from OR combines the merits of both approaches. We introduce a general way of how linear optimization constraints can strengthen their propagation abilities via Lagrangian relaxation. The method is evaluated on a set of benchmark problems stemming from a multimedia application. The experiments show the superiority of the combined method compared with a pure OR approach and an algorithm based on two independent optimization constraints.

Keywords: optimization constraint, maximum weighted stable set constraint, knapsack constraint, Lagrangian relaxation

1 Introduction

One important branch concerning the integration of CP and OR is the development of *optimization constraints* [9]. Given a maximization problem $P(x)$, the idea is to use upper bound information to detect non-profitable assignments for a variable: If an upper bound on $P(x_{|x_i=k})$ drops below the best known solution value, then k can be removed from the domain of x_i. Filtering algorithms based on this idea are of great importance when combinatorial optimization problems have to be solved exactly. For many applications, the tightening of problem formulations within a branch-and-bound approach has shown to improve on the quality of the bounds computed as well as on the approach's robustness.

Frequently, optimization problems can be viewed as a combination of two or more simpler structured problems. Assuming that efficient propagation algorithms (PAs) for their corresponding optimization constraints exist, their independent application usually does not yield an effective PA for the combined constraint. The reason for this is

* This work was partly supported by the German Science Foundation (DFG) project SFB-376, by the UP-TV project, partially funded by the IST program of the Commission of the European Union as project number 1999-20 751, and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

that tight bounds on the objective cannot be obtained when some restrictions are ignored. Therefore, the standard CP method to have constraints collaborate via sequential domain reductions is not suitable for optimization constraints.

On the other hand, in mathematical programming *Lagrangian* and *Surrogate relaxations* have been developed to compute tight bounds on the objective while maintaining the desired substructures. We show how to use Lagrangian relaxation for the efficient coupling of optimization constraints.

For a multimedia application incorporating a knapsack problem (KP) and a maximum weighted stable set problem (MWSSP) on an interval graph, we show exemplary how linear optimization constraints can be coupled via Lagrangian relaxation to achieve an effective PA for the combined optimization constraint.

1.1 Previous Work

Optimization constraints and the concept of cost based filtering are presented in [18]. The paper also provided a recent overview on attempts of combining methods from CP and OR.

The work presented here is particularly based on some previous results on specific optimization constraints: In [7, 14], an efficient shortest path constraint was introduced. It is based on the computation of shortest paths in directed acyclic graphs [1], and achieves bound consistency in time $O(n+a)$ on DAGs with n nodes and a arcs. In [20], the special case of shortest path constraints in co-interval graphs is considered. It was shown that finding shortest paths in node weighted co-interval graphs solves the maximum weighted stable set problem on interval graphs, and a propagation algorithm with amortized linear time complexity was developed.
Knapsack constraints have been studied in [8]. There, amortized linear time PAs based on continuous bounds were developed.

Lagrangian relaxation was first presented in [6] for resource allocation problems. Held and Karp used it for the TSP [11, 12], and it has been applied in many different areas since then. We refer to [1] for an introduction.
In [9], Focacci et al. introduced a method to strengthen bound propagation using Lagrangian relaxation. In that approach, Lagrangian multipliers were used to incorporate additional cuts to tighten the bound used for propagation.

The multimedia application we use to introduce CP-based Lagrangian Relaxation is further investigated in the EU-project “UP-TV” where partners from industry and academia work together on open problems in the area of digital broadcasting. A first approach on an extended version of the problem considered here is described in [16].

We present in detail how the coupling of linear optimization constraints can be performed efficiently using Lagrangian relaxations. Some parts of this paper have already been published in [20].

The paper is structured as follows: In Section 2, we formally introduce the idea of CP based Lagrangian Relaxation and some related special aspects. To give a concrete application of the method, in Section 3 we introduce the Automatic Recording Problem (ARP), that can be viewed as a combination of a knapsack problem and a MWSSP on interval graphs. Then, in Section 4 the concept of CP based Lagrangian Relaxation is applied on the ARP. Finally, in Section 5 we give numerical results comparing different PAs for the ARP optimization constraint.

2 Coupling Propagation Algorithms via Lagrangian Relaxation

We consider an integer linear optimization problem (P) consisting of the two constraint families $\mathcal{A}$: $Ax \leq b$, $x \in \mathbb{N}^n$, and $\mathcal{B}$: $Bx \leq d$, $x \in \mathbb{N}^n$:

$$\begin{array}{rl} (P): \text{Maximize} & p^T x \\ \text{subject to} & Ax \leq b \\ & Bx \leq d \\ & x \in \mathbb{N}^n \end{array}$$

Further, we assume that PAs Prop($\mathcal{A}$) and Prop($\mathcal{B}$) exist for each of the two families. Then, an obvious approach to solve P exactly is to apply a branch-and-bound algorithm using linear relaxation bounds for pruning and the existing PAs Prop($\mathcal{A}$) and Prop($\mathcal{B}$) to tighten the problem formulation in every choice point.

Even though Prop($\mathcal{A}$) and Prop($\mathcal{B}$) may be efficient and effective for the substructures they have been designed for, their application for the combined problem is usually not. This is because tight bounds on the objective cannot be obtained by taking only a subset of the restrictions into account. An accurate bound on the overall problem can only be computed by looking at the entire problem, i.e., it cannot be achieved by looking at either one constraint family only.

Linear relaxation bounds can easily be obtained by applying a standard LP solver. That bound often yields a good estimate on the performance that can (still) be reached. However, it is not straight forward to see how it could be exploited for efficient propagation. Applying conventional reduced cost propagation only indirectly exploits the structure of the problem and therefore appears ineffective, whereas to perform probing via full reoptimization can be very costly and thus inefficient.

2.1 Constraint Coupling

Lagrangian relaxation allows us to bring together the advantages of a tight continuous global bound and the existing PAs that exploit the special structure of their respective constraint families. We introduce non-negative Lagrange multipliers $\lambda_i \geq 0$ and define the Lagrangian subproblem

$$\begin{array}{rl} L_{\mathcal{B}}(\lambda): \text{Maximize} & z(\lambda) := p^T x - \lambda^T (Ax - b) \\ \text{subject to} & Bx \leq d \\ & x \in \mathbb{N}^n \end{array}$$

Then, the Lagrange multiplier problem is to solve

$$\begin{array}{rl} \text{Minimize} & z(\lambda) \\ \text{subject to} & \lambda \geq 0 \end{array}$$

For every $\lambda \geq 0$, $z(\lambda)$ is a valid upper bound on the objective. Therefore, we can apply Prop($\mathcal{B}$) on the constraint family $\mathcal{B}$ each time we solve the Lagrangian subproblem $L_{\mathcal{B}}(\lambda)$. After we have found optimal Lagrange multipliers λ^*, i.e. $z(\lambda^*) \leq z(\lambda)\ \forall\ \lambda \geq 0$, we use the (optimal) dual information π on the constraint family $\mathcal{B}$ to perform cost based filtering with respect to that substructure. By Lagrange relaxing the second constraint family with multipliers $\pi \geq 0$, we obtain:

$$
\begin{aligned}
L_{\mathcal{A}}(\pi) : \text{Maximize} \quad & p^T x - \pi^T(Bx - d) \\
\text{subject to} \quad & Ax \leq b \\
& x \in \mathbb{N}^n
\end{aligned}
$$

To summarize: two linear optimization constraint families $\mathcal{A}$ and $\mathcal{B}$ for which efficient PAs Prop($\mathcal{A}$) and Prop($\mathcal{B}$) are known can be combined effectively by computing Lagrangian multipliers for $\mathcal{A}$, using Prop($\mathcal{B}$) for propagation in each Lagrangian subproblem $L_{\mathcal{B}}(\lambda)$, and then handing back dual information of the optimal $L_{\mathcal{B}}(\lambda^*)$ on $\mathcal{B}$ to propagate $\mathcal{A}$ with the corresponding (optimal) reduced cost objective, i.e. we apply Prop($\mathcal{A}$) on $L_{\mathcal{A}}(\pi)$. This procedure even strengthens the bound on the objective, as domain reduction is also performed during the bound computation. However, there arises one problem: if domains are reduced during the process of finding optimal Lagrange multipliers, the algorithm that solves the Lagrangian dual must be aware of this. It is subject to further research, how e.g. subgradient methods must be adapted to be able to cope with that situation.

2.2 Generalizations

Before applying the concept of CP based Lagrangian Relaxation to a multimedia application, we present some extensions and generalizations of the concept presented above.

Coupling more than Two Optimization Constraints The procedure sketched can easily be generalized if the coupling of more than two constraints is desired. All we need to do is select the substructure that determines the Lagrangian subproblem, i.e., the one that has to be solved several times with changing objective functions. After we have computed (near) optimal Lagrange multipliers, we apply the propagation algorithm for the other substructures with a modified objective function. That modification is determined by the dual values of the family of constraints in the Lagrangian subproblem and the Lagrange multipliers for the remaining substructures.

Linear Relaxations If continuous bounds are preferred to bounds based on Lagrangian relaxations, it is also possible to use dual values instead of Lagrange multipliers to modify the objective functions for the respective subproblems we want to apply a PA on. We still use the terminology of a coupling method based on Lagrangian relaxation, as we use Lagrangian objectives for cost based filtering.

Notice, that the method can also be used in combination with tightening algorithms such as cut generators. We simply incorporate all additional cuts as a new family of constraints we have to find Lagrange multipliers (or dual values) for.

Binary IPs Interestingly, as a special case we achieve a propagation algorithm for binary IPs. Given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $p \in \mathbb{R}^n$, we consider the following binary program:

$$\begin{array}{ll} \text{Maximize } p^T x & \\ \text{subject to} \quad Ax \leq b & \\ \qquad\qquad\quad x \in \{0,1\}^n & \end{array}$$

The problem can be viewed as a combination of m knapsack problems. Assuming that we solve the continuous relaxation to compute an upper bound, let $\pi \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n$ denote the optimal solution to the dual problem, i.e., π and μ solve the following linear problem:

$$\begin{array}{ll} \text{Minimize } b^T\pi + 1^T\mu & \\ \text{subject to} \quad A^T\pi + \mu \geq p & \\ \qquad\qquad\qquad\quad \pi, \mu \geq 0 & \end{array}$$

Let $0 \leq i < m$, and let ${}^iA \in \mathbb{R}^{(m-1)\times n}$ denote the matrix that evolves from A by erasing row i, and ${}^ib$, ${}^i\pi \in \mathbb{R}^{m-1}$ the vectors that evolve by erasing component i from b and π, respectively. Further, let a_i denote the ith row of matrix A. Then, for every $0 \leq i < m$ we perform domain reduction with respect to the following knapsack problem:

$$\begin{array}{ll} \text{Maximize } (p^T - {}^i\pi^T\,{}^iA)x + {}^i\pi^T\,{}^ib & \\ \text{subject to} \qquad\qquad\qquad\quad a_i x \leq b_i & \\ \qquad\qquad\qquad\qquad\qquad\quad x \in \{0,1\}^n & \end{array}$$

Thus, as a special application of CP based Lagrangian Relaxation, we achieve an effective filtering algorithm for binary IPs that runs in $\Theta(mn \log n)$ (using [8]) after we have found optimal dual values of the continuous relaxation.

3 The Automatic Recording Problem

In the following, we introduce a multimedia application for which we tested the method of coupling PAs via Lagrangian relaxation.

The *Automatic Recording Problem (ARP)* is an example of a problem that is constituted by two simpler constraints. We focus on algorithms that solve the problem exactly and give a tightened formulation of the ARP as an integer program (IP).

The technology of digital television offers new possibilities for individualized services that cannot be provided by nowadays analog broadcasts. Additional information like classification of content, or starting and ending times can be submitted within the digital broadcast stream. With those informations at hand, new services can be provided that make use of individual profiles and maximize customer satisfaction.

One service – which is available already today – is an "intelligent" digital video recorder that is aware of its users' preferences and records automatically [2]. Such a recorder tries to match a given user profile with the information submitted by the different TV channels. E.g., a user may be interested in thrillers, the more recent the better.

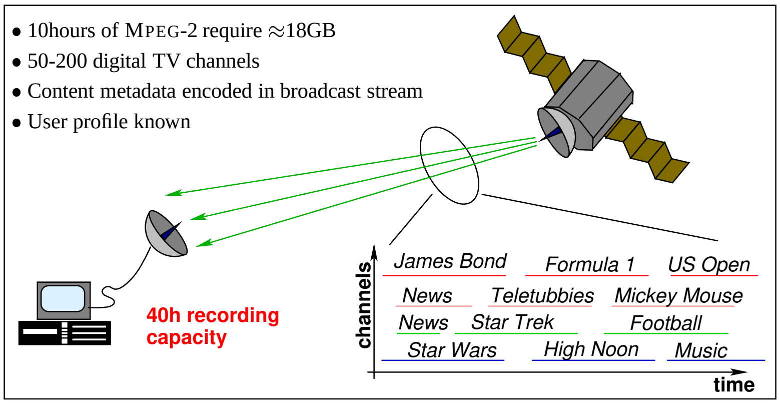

Fig. 1. The scenario for the Automatic Recording Problem

The digital video recorder is supposed to record programs such that the users' satisfaction is maximized. As the number of channels may be enormous (more than 100 digital channels are possible), a service that automatically provides an individual selection is highly appreciated and subject of current research activities (for example within projects like *UP-TV* funded by the European Union or the *TV-Anytime Forum*).

In this context, two restrictions have to be met. First, the storage capacity is limited (10hours of MPEG-2 video need about 18 GB). And second, only one video can be recorded at a time. (see Fig. 1).

More formally, we define the problem as follows:

Definition 1. *Let* $n \in \mathbb{N}$, $V = \{0, \ldots, n-1\}$ *the set of programs,* $start(i) < end(i)\ \forall\ i \in V$ *the corresponding starting and ending times.* $w = (w_i)_{0 \le i < n} \in \mathbb{R}_+^n$ *the storage requirements,* $K \in \mathbb{R}_+$ *the storage capacity, and* $p = (p_i)_{0 \le i < n} \in \mathbb{N}^n$ *the profit vector.*

We say that the interval $I_i := [start(i), end(i)]$ corresponds *to program* $i \in V$*, and call two programs* $i, j \in V$ overlapping *whose corresponding intervals overlap, i.e.* $I_i \cap I_j \neq \emptyset$*. For* $X \subseteq V$ *we call* $p_X := \sum_{i \in X} p_i$ *the* user satisfaction (with respect to X).

The ***Automatic Recording Problem (ARP)*** *then is to find a subset* $X \subseteq V$ *such that*

(a) X *can be stored within the given disc size, i.e.* $\sum_{i \in X} w_i \le K$.
(b) at most one program must be recorded at a time, i.e. $I_i \cap I_j = \emptyset\ \forall\ i \neq j \in X$
(c) X *maximizes the user satisfaction, i.e.* $p_X \ge p_Y\ \forall\ Y \subseteq V$, Y *respecting (a) and (b).*

3.1 On the Complexity of the Automatic Recording Problem

Obviously, even if all programs are pairwise non-overlapping (i.e., if restriction (b) is obsolete), it remains to solve a knapsack problem. Thus, the ARP is NP-hard. Let

$p_{max} := \max\{p_i \mid 0 \leq i < n\}$. 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.

A Dynamic Programming Algorithm The algorithm we develop in the following is similar to the teaching book dynamic programming algorithm for knapsack problems. Setting $\overline{\mathbb{R}} := \mathbb{R} \cup \{\infty\}$ and $\psi := np_{max} + 1$, we compute a matrix $M = (m_{kl}) \in \overline{\mathbb{R}}^{n \times \psi}$, $0 \leq k < \psi$, $0 \leq l < n$. 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 $\sum_{0 \leq i \leq l} p_i < k$).

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

$$e_{last_j} < s_j \text{ and } e_i \geq s_j \quad \forall\, last_j < i \leq j.$$

We set $last_j := -1$ iff no such node exists, i.e., iff $e_0 \geq s_j$. To simplify the notation, let us assume that $m_{k,-1} = \infty$ for all $0 < k < \psi$, and $m_{k,-1} = 0$ for all $k \leq 0$. Then,

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

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 $0 \leq i < n$. Both can be done in time $\Theta(n \log n)$. Then, we build up the matrix row by row. Finally, we compute $\max\{k \mid m_{k,n} \leq K\}$. The total running time of this procedure and the memory needed are obviously in $\Theta(n^2 p_{max})$.

A Fully Polynomial Time Approximation Scheme (FPTAS) As for knapsack problems, we can use the dynamic programming algorithm to derive an FPTAS by scaling the profit vector. Given $\varepsilon > 0$, we set $S := \varepsilon p_{max}/n$, and $\overline{p_i} := \lfloor p_i/S \rfloor$. Then, $\overline{p_{max}} = \lfloor n/\varepsilon \rfloor$. Thus, the running time of our dynamic programming algorithm applied with the scaled profit vector is in $\Theta(n^3/\varepsilon)$.

Next, we study the error we make by using $\overline{p}$ instead of p. Let $x \in \{0,1\}^n$ denote an optimal solution with respect to p, and $\overline{x} \in \{0,1\}^n$ an optimal solution with respect to $\overline{p}$. Then,

$$\begin{array}{ccccccc} p^T\overline{x} & \geq & S\lfloor p^T/S \rfloor \overline{x} & = & S\overline{p}^T\overline{x} & \geq & S\overline{p}^T x \\ & \geq & S((p^T x)/S - n) & = & p^T x - Sn. & & \end{array} \tag{1}$$

Therefore,

$$|p^T x - p^T \overline{x}|/p^T x \quad \leq \quad Sn/p_{max} \quad = \quad \varepsilon,$$

i.e., the relative error is at most ε. Thus, we have found an FPTAS for the ARP.

3.2 A Mathematical Programming Formulation

Because the problem of finding and proving optimal solutions is of interest in its own right, and also because the FPTAS we developed is far too memory consuming to be of practical relevance, we focus on exact approaches to solve the ARP. Using mathematical programming, the problem can be stated as an integer program (IP):

$$\begin{array}{lrl}\text{Maximize} & \sum_{0\leq i<n} p_i x_i & \\ \text{subject to} & x_i + x_j \leq 1 & \forall\, 0 \leq i < j < n,\ I_i \cap I_j \neq \emptyset \\ & \sum_{0\leq i<n} w_i x_i \leq K & \\ & x \in \{0,1\}^n & \end{array} \qquad (IP\ 1)$$

The objective function maximizes the user satisfaction. Constraints of the form $x_i + x_j \leq 1$ ensure that for overlapping intervals I_i, I_j at most one program can be selected. Storage restrictions are enforced by the last row. The formulation can be tightened by replacing the overlapping constraints with maximal clique constraints.

Definition 2. *A set $C \subseteq V$ is called a* conflict clique*, iff $I_i \cap I_j \neq \emptyset\ \forall\ i, j \in C$. A conflict clique C is called* maximal*, iff $\forall\ D \subseteq V$, D conflict clique: $C \subseteq D \Rightarrow C = D$. Let $M := \{C_0, \ldots, C_{m-1}\} \subseteq 2^V$ the set of maximal conflict cliques.*

Then, restrictions of the form $\sum_{i\in C_p} x_i \leq 1\ \forall\ 0 \leq p < m$ imply that $x_i + x_j \leq 1$ for all nodes $i, j \in V$ whose corresponding intervals overlap. On the other hand, if $x_i + x_j \leq 1$ for all overlapping intervals, it is also true that $\sum_{i\in C_p} x_i \leq 1\ \forall\ 0 \leq p < m$. Thus, IP (1) is equivalent to

$$\begin{array}{lrl}\text{Maximize} & \sum_{0\leq i<n} p_i x_i & \\ \text{subject to} & \sum_{i\in C_p} x_i \leq 1 & \forall\, 0 \leq p < m \\ & \sum_{0\leq i<n} w_i x_i \leq K & \\ & x \in \{0,1\}^n & \end{array} \qquad (IP\ 2)$$

Though being NP-complete on general graphs, finding maximal cliques on the graph defined by our application is simple. Since we can record two programs only if they do not overlap in time, the graph defined by the broadcast does contain arcs from node i to node j iff $I_i \cap I_j = \emptyset$. Complements of theses graphs are known as interval graphs:

Definition 3. *A graph $G = (V, E)$ is called an* interval graph *if there exist intervals $I_1, \ldots, I_{|V|} \subset \mathbb{R}$ such that $\forall\ v_i, v_j \in V : \{v_i, v_j\} \in E \iff I_i \cap I_j \neq \emptyset$.*

Many hard problems are rather easy to solve on these graphs. In particular, the computation of maximal cliques can be performed in time $\Theta(n \log n)$ on interval graphs [10]. Hence, (IP 2) can be obtained in polynomial time.

3.3 Solving the Resulting Integer Program

There exists quite a bunch of methods applicable for the ARP. We have mentioned already a dynamic programming approach in sect. 3.1. Also, using labeling algorithms for constrained shortest paths [5, 4] seems to be a good choice. In the presents of additional constraints, those methods often fail since they are fixed to a particular structure. We therefore solve the ARP via a constraint programming based branch-and-bound approach that enables us to add (real-life) constraints to the problem without changing the core ideas.

Branch-and-bound approaches have proofed to be efficient, widely applicable and thus are quite commonly used. In every choice point, a bound based on some (often continuous) relaxation is being computed. If that bound is worse than the objective value B of the incumbent solution, backtracking occurs. A successful application of the branch-and-bound paradigm relies heavily on tight bounds that can be computed quickly. Domain reduction can help to improve on the performance of a branch-and-bound search if the filtering is both, effective and efficient. Effective means, that it must have an impact, i.e., it has to be able to filter many values, whereas the efficiency measures how quickly the routine works.

The effectivity of a filtering algorithm mainly depends on the quality of bounds it uses to estimate the impact of fixing a variable to one of its values. For the ARP, our experiments show that the continuous relaxation bound yields a good estimate on the solution quality that can be reached. Thus, it can be used for pruning purposes in a branch-and-bound approach. But it is not straight forward to see, how this bound could be used for filtering purposes effectively, that is, other than by probing via full reoptimization, which is inefficient. On the other hand, domain reductions with respect to reduced cost information can be done quickly, but is not very effective.

4 CP based Lagrangian Relaxation for the ARP

The ARP can be viewed as a combination of two simpler optimization constraints: a knapsack constraint, and a maximum weighted stable set constraint on an interval graph. For the knapsack constraint, a propagation algorithm was already developed in [8]. It runs in time $\Theta(n \log n)$, and in amortized linear time for $\Omega(\log n)$ choice points. In [20], we developed a PA for the maximum weighted stable set substructure (MWSSP) of the ARP. The algorithm runs in time $\Theta(n \log n)$ or in amortized linear time for $\Omega(\log n)$ incremental propagation calls.

With the two PAs at hand, we are able to perform domain reduction for the two natural substructures of the ARP. According to the abstract description in Section 2, we will now tie the two filtering algorithms together:

As the PA for MWSSP allows us to incorporate changing objectives at a low computational cost, we decide to relax the capacity constraint. We introduce a non-negative Lagrange multiplier $\lambda \geq 0$ and define the Lagrangian subproblem

$$\begin{array}{lll} \text{Maximize} & z(\lambda) := z & L(\lambda) \\ \text{subject to} & z = \sum_{0 \leq i < n} (p_i - \lambda w_i) x_i + \lambda K & \\ & \sum_{i \in C_p} x_i \leq 1 & \forall\, 0 \leq p < m \\ & x \in \{0,1\}^n & \end{array}$$

The Lagrange multiplier problem then is to solve Minimize $z(\lambda)$, such that $\lambda \geq 0$. For every $\lambda \geq 0$, $z(\lambda)$ is a valid upper bound on the objective. Therefore, we can apply cost based filtering for the MWSS constraint on interval graphs each time we solve the Lagrangian subproblem. After we have found an optimal Lagrange multiplier λ^*, we use dual information $\pi \in \mathbb{R}^m$ from the corresponding stable set subproblem to perform variable fixing with respect to the knapsack substructure now. By Lagrange relaxing the maximal clique constraints with multipliers $\pi \geq 0$, we obtain a knapsack problem. Let $\mu_i := \sum_{j\,:\,i \in C_j} \pi_j \ \forall\, 0 \leq i < n$ and $\overline{\pi} := \sum_{0 \leq j < m} \pi_j$. The problem then is to

$$\begin{array}{ll} \text{Maximize} & \sum_{0 \leq i < n} (p_i + \mu_i) x_i - \overline{\pi} \\ \text{subject to} & \sum_{0 \leq i < n} w_i x_i \leq K \\ & x \in \{0,1\}^n \end{array}$$

Relaxations of this problem again yield a valid upper bound, and we can propagate the knapsack optimization constraint on the modified objective.

4.1 Implementation Details

As we mainly focus on the general principle of how existing PAs can be exploited more effectively by tying them together using Lagrangian relaxation, so far we have left out some implementation details concerning the choice of the branching variable and the computation of the Lagrange multiplier λ^*. In this section, we will give an insight in the implementation the tests were performed with.

We used four different approaches for our experiments: the first is a pure branch-and-bound algorithm without any problem tightening (referred to as *P-0*). The second uses the PAs for knapsack and maximum weighted stable set problems on the original objective (*P-1*). The third and the fourth approach (*P-2* and *P-3*) realize the idea of coupling PAs for linear optimization constraints via Lagrangian relaxation. *P-2* calls for domain reduction just once after the Lagrangian dual has been solved, whereas *P-3* also propagates the maximum weighted stable set constraint during the search for optimal Lagrange multipliers.

Continuous Bound Computation For pruning, the computation of a linear bound on the objective is needed. *P-2* and *P-3* obviously use the objective value corresponding to $L_{\mathcal{B}}(\lambda^*)$ for this purpose. As the computation via Lagrangian relaxation with stable set subproblems turned out to be very efficient, we used that algorithm for all four approaches.

Computation of $\boldsymbol{\lambda}^*$ To determine λ^*, we used a method to maximize one-dimensional concave functions based on the golden section. We obtain a sequence of λ^k, $k \in \mathbb{N}$. Let $e_{max} := \max\{p_i/w_i \mid 0 \leq i < n\}$. Then, for all $\varepsilon > 0$ there exists a constant $c > 0$ such that

$$|\lambda^k - \lambda^*| < \varepsilon \qquad \forall\, k \geq c \cdot \log e_{max}$$

Thus, after $O(\log e_{max})$ iterations we can numerically approximate the optimal Lagrange multiplier λ^*. Each iteration costs amortized linear time for at least $\Omega(\log n)$

choice points. Finally, in every choice point we add $O(n \log n)$ for the succeeding knapsack PA. Therefore, the coupled PA for the tight global continuous relaxation bound runs in time $O(n \log e_{max} + n \log n)$.

Notice, that the Lagrangian subproblem is totally unimodular. Thus, the Lagrangian relaxation bound has the same value as the bound that evolves from a linear continuous relaxation.

Branching Variable Selection All algorithms choose the first node on the shortest path[1] with maximal efficiency p_i/w_i as branching variable.

5 Numerical Results

All experiments were performed on a PC with an AMD-Athlon 600 MHz processor and 256 MB RAM running Linux 2.2. The implementation was done in C++ and compiled by gcc 2.95 with maximal optimization (O3). The algorithms were built on top of ILOG SOLVER 5.0 [15].

5.1 Test Instance Generation

The experiments were conducted on several sets of randomly generated test instances. To achieve scenarios which we believe to be of relevance for the real life application, each set of instances is generated by specifying the time horizon (half a day to 3 days) and the number of channels (20 – 100). 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. We consider either 3, 5, or 7 different classes, respectively. The lengths of programs in the classes vary from 5±2 minutes to 150±50 minutes. The disc 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. For the experiments, we used four different strategies for the computation of an objective function:

- 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 correlated (TC)* 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 *weakly correlated (TWC)* instances, that value is perturbed by a noise of ±20%.
- In the *subset sum (SSS)* data finally, the profit of a program simply equals its length.

[1] according to the optimal reduced costs objective $\sum_{0 \leq i < n}(p_i - \lambda^* w_i)x_i + \lambda^* K$

The different objectives try to emulate some effects we believe to hold real life instances. In the CU instances for example, programs of the same class cause similar attractions. And the TC and TWC instances cause many conflicts regarding the choice of programs that are being broadcasted at the same time. However, the different strategies we considered are only intuitively justified. The feasibility of our approach for real life instances can only be concluded from the fact, that we achieve similar results for all choices of the objective.

We identify a test set by giving the parameters the generator was started with. According to the above description those parameters are:

- time horizon in minutes,
- number of channels,
- number of different classes [3, 5, or 7], and
- objective type [CU, TC, TWC, or SSS].

5.2 Experimental Evaluation

In the following we present some numerical results. The experiments consist of 50 random instances per test set. For each instance, the approaches *P-0* – *P-3* were run to find and prove an optimal solution. We protocol running times and the number of choice points needed for an exhaustive search. All approaches find a first solution rather early in the search. Therefore, the main work lies in the proof of optimality rather than in the construction of the solution. We conclude that the branching variable selection we used efficiently supports finding near-optimal solutions in a non-exhaustive search.

Table 1 shows the performance (time and choice points) of all four approaches on test sets generated with a time horizon of 12 hours and 20 channels using 5 different program classes and CU, TC, and TWC to determine the objective function.

test set	*P-0*		*P-1*		*P-2*		*P-3*	
5 . . .	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*
CU	9.5	519.4	5.2	295.5	7.0	198.5	8.6	184.0
TC	441.8	40155.7	67.2	4525.8	18.0	696.5	28.4	575.6
TWC	15.5	1136.1	12.0	802.6	6.2	339.9	9.5	321.2

Table 1. Test set with 5 classes, 12 hours, 20channels and different objectives. Times in seconds and choice points are averages for 50 randomly generated instances. The average number of programs per instance are between 607.6 and 612.6.

When comparing the different types of objectives, we find that for all four approaches the TC instances are much harder than CU and TWC, which are comparably difficult to solve. This is a general observation we made for all kinds of different test sets using 3 or 7 classes as well as different time horizons and numbers of channels.

We further observe that a higher degree of coupling between the two optimization constraints yields a partially drastic reduction in the number of choice points of up to a factor of about 70 on the difficult time correlated instances. Regarding the computation time, there is a trade off between the reduction of choice points and the time spent per choice point(TpCP). The TC and TWC instances show, that *P-2* can outperform *P-3* because of the shorter TpCP that is needed for that degree of integration. When comparing *P-1* and *P-3* on the CU instances, the reduction of choice points is not big enough to justify the bigger TpCP needed, and *P-1* is the approach that takes the least computation time.

Generally, a bigger reduction of choice points is more likely to pay off when the absolute TpCP needed is rather high. Particularly, this holds for applications where additional constraints are propagated on top of the objective constraint itself. For the ARP, the optimization constraint is the only active constraint. Therefore, to justify the worse TpCP caused by the more complicated propagation algorithm, a noteworthy reduction of the number of choice points must be achieved. The *P-2* and *P-3* approaches obtain a sufficient reduction of choice points on the more difficult TC test sets, and also for larger test instances:

Table 2 shows the performances of all approaches on test instances that were generated using 5 different program classes with different time horizons and numbers of channels. The objective was computed according to the chosen classes, i.e., according to CU. As expected, for the larger instances with a time horizon of 72 hours (3 days) and 20 channels, the two coupling approaches *P-2* and *P-3* outperform *P-1* roughly by a factor of 4 regarding the number of choice points and almost a factor of 2 with respect to the computation time needed. The minima, maxima and standard deviations proof that the average numbers we present are not biased by very few outliers, but represent meaningful values for the evaluation of the algorithms performances.

In Table 3 we compare the different approaches on a whole variety of test instances that were generated using very different parameters and objective functions. Again, relevant and partially huge reductions in the number of choice points can be obtained by CP based Lagrangian Relaxation realized in *P-2* and *P-3*.

So far, we have left out comparisons regarding the choice of the objective according to SSS. Table 4 shows the results obtained for a collection of very different test sets generated with SSS. Two facts stand out: first, a comparison with Table 1 shows, that the SSS instances are much easier to solve than for other choices of the objective. And second, *P-1* achieves only a slight reduction of choice points compared to *P-0* that cannot be improved by *P-2* and *P-3* at all.

The effect is not surprising: We considered the somewhat artificial SSS test sets because of their obvious relation to subset sum benchmarks for knapsack problems. Because of the equal efficiency p_i/w_i of all programs, the knapsack optimization constraint has great difficulty to include or exclude programs. Thus, that constraint is not effective, and the burden of domain reduction lies on the MWSSP optimization constraint only. In total, using the optimization constraint for pruning purposes only is most time efficient here.

Finally, we investigate the impact of the number of channels. Table 5 shows a comparison of three different test sets that were generated using 3 different program classes

test set		*P-0*		*P-1*		*P-2*		*P-3*	
5 CU		*time*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*
	avg	*2.4*	*238.3*	*1.3*	*129.9*	*1.4*	*108.6*	*2.1*	*89.9*
12h	min	0.1	5.0	0.1	5.0	0.1	5.0	0.1	5.0
20ch	max	25.4	2216.0	15.1	1531.0	12.9	1269.0	24.2	1045.0
	std	4.2	396.8	2.3	243.6	2.5	206.5	4.0	173.5
	avg	*16.5*	*741.9*	*8.2*	*370.1*	*9.9*	*272.0*	*14.2*	*250.5*
12h	min	0.1	3.0	0.2	3.0	0.2	3.0	0.2	3.0
50ch	max	167.3	7058.0	82.6	3615.0	154.1	2664.0	156.5	2664.0
	std	30.8	1377.6	14.5	658.9	22.9	506.9	26.8	465.0
	avg	*9.5*	*519.4*	*5.2*	*295.5*	*7.0*	*198.5*	*8.6*	*184.0*
24h	min	0.5	21.0	0.4	13.0	0.3	10.0	0.5	10.0
20ch	max	87.5	4416.0	59.6	3762.0	93.4	2094.0	98.1	2067.0
	std	15.2	829.9	9.3	587.6	17.2	377.8	17.7	374.9
	avg	*1104.9*	*24301.4*	*585.2*	*14219.3*	*883.3*	*8371.9*	*921.5*	*8286.8*
24h	min	0.8	12.0	1.0	12.0	0.7	9.0	1.1	9.0
50ch	max	31045.5	675235.0	15625.2	368440.0	33281.3	292753.0	31573.5	292753.0
	std	4448.4	97121.0	2272.6	54288.6	4662.0	41139.4	4441.2	41121.2
	avg	*2627.7*	*40901.5*	*1786.7*	*27662.0*	*920.4*	*6674.7*	*990.9*	*6514.7*
72h	min	2.0	29.0	2.4	29.0	3.0	29.0	5.5	29.0
20ch	max	32751.9	460350.0	30520.3	412421.0	11766.0	90397.0	13724.7	89589.0
	std	5514.7	85325.8	4543.1	65188.4	1996.8	14515.9	2189.7	14379.4

Table 2. Test set with 5 classes, objective: CU for various time horizons (in hours) and channel numbers (ch). Italic number give the average (avg) time and nodes, resp. of 50 instances. Numbers below are: minimum (min), maximum (max), and standard deviation (std) for these 50 instances. The average number of programs per instance are 315.2 for (12h/20ch), 793.5 (12h/50ch), 607.6 (24h/20ch), 1512.1 (24h/50ch), and 1782.6 (72h/20ch), resp.

	P-0		*P-1*		*P-2*		*P-3*	
test set	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*	*time*	*nodes*
3 CU 120h 20ch	5210.3	60839.2	1734.4	30676.4	455.9	3433.9	490.1	2945.1
5 TWC 72h 20ch	11600.1	293386.8	1526.8	35718.4	261.0	3683.6	411.7	3134.5
7 TC 24h 50ch	8349.0	250367.1	4066.3	105572.6	403.4	6235.4	533.0	4219.1

Table 3. Effectivity of the different approaches for different benchmark classes, we have an average of 1956.7 programs for the 120h/20ch test, 1782.6 for 72h/20ch, and 1423.3 for 24h/50ch

test set 5 SSS		P-0 time	P-0 nodes	P-1 time	P-1 nodes	P-2 time	P-2 nodes	P-3 time	P-3 nodes
12h 20ch	316.4p	0.2	23.1	0.2	15.2	0.2	15.2	0.3	15.2
12h 50ch	792.4p	0.5	18.8	0.5	13.9	0.5	13.9	0.8	13.9
24h 20ch	611.2p	0.5	26.9	0.6	21.9	0.6	21.9	1.0	21.9
24h 50ch	1527.3p	1.6	29.1	1.8	23.2	1.7	23.2	3.0	23.2
72h 20ch	1778.3p	3.5	53.4	4.6	51.7	5.0	51.7	8.7	51.7
72h 50ch	4464.3p	11.0	54.4	14.4	52.8	15.4	52.8	27.2	52.8

Table 4. Subset sum data sets for 12 hours, 20 channels, upto 72 hours and 50 channels. The average number of programs is given as parameter p

and CU objectives. All instances have a similar size and roughly contain about 1000 programs. We observe that the instances become more difficult to solve for all approaches when the number of channels is decreasing. This surprising result may be caused by the fact that a smaller number of channels increases the relative importance of the knapsack optimization constraint that is more difficult than the MWSSP constraint. However, a sound answer to that question can only be given by profound further investigations. It remains to note that for TC data sets we observed a converse behavior: the instances become more difficult the more channels are involved which is obviously caused by many temporally conflicting programs of similar value.

test set 3 CU	*avg. no of programs*	*P-0 time*	*P-0 nodes*	*P-1 time*	*P-1 nodes*	*P-2 time*	*P-2 nodes*	*P-3 time*	*P-3 nodes*
12h 100ch	1048.3	18.8	680.8	9.1	326.3	5.1	108.6	7.6	95.5
24h 50ch	1013.4	37.8	1461.1	19.5	734.4	11.1	187.9	13.8	178.5
72h 20ch	1175.0	177.4	3003.1	111.7	1897.2	36.6	468.4	42.9	401.2

Table 5. Comparing different benchmark sets about 1000 programs for 3 classes and objective CU

6 Conclusions and Future Work

For the automatic recording problem, we exemplary introduced the idea of coupling propagation algorithms (PAs) via Lagrangian relaxation. It allows to combine existing propagation algorithms for linear optimization constraints to obtain effective and efficient filtering algorithms based on tight global bounds. We believe, that this idea is generic and independent of the specific application we presented to base an empirical evaluation on. The numerical results show a significant improvement due to the coupling method with respect to the computation time and the number of choice points.

The method is suited for linear optimization problems for which bounds based on continuous or Lagrangian relaxations can be used effectively.

For the multimedia application we introduced, we developed an FPTAS and formulated a refined IP formulation. The continuous relaxation of that IP yields a tight upper bound as our experiments showed. Several extensions are possible for that application. A digital video recorder could have more than one recording unit which allows the recording of a limited number of channels simultaneously. In an IP context, this modification can be introduced easily. For the exact approach presented, a fast and efficient PA for this type of relaxed non-overlapping constraint is subject to further research.

Finally, as an open topic it remains to investigate, how general algorithms for the solution of the Lagrangian dual (such as subgradient or multiplier adjustment algorithms) must be adapted to allow domain reductions during the search for optimal Lagrange multipliers. Taking subgradient algorithms as an example: Is it really necessary to reset the iteration limit and the step length after every domain reduction that has taken place, or can convergence still be proved for more optimistic strategies?

References

1. R.K. Ahuja, T.L. Magnati, J.B. Orlin. *Network Flows.* Prentice Hall, 1993.
2. *mediaTV*, technical description, Axcent AG. `http://www.axcent.de`.
3. E. Balas and E. Zemel. An algorithm for large-scale zero-one knapsack problems. *Operations Research*, 28:119–148, 1980.
4. J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In Ball, Magnanti, Monma, Nemhauser (editors): Network Routing. *Handbooks in Operations Research and Management Science*, Vol. 8, pp. 35–139, North-Holland, 1995.
5. J. Desrochers and F. Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFORS*, 26(3), p. 191–211, 1988.
6. H. Everett. Generalized lagrange multiplier method for solving problems of optimum allocation of resource. *Operations Research* 11:399–417, 1963.
7. T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. Journal of Heuristics, Vol 8(1), p. 59–81, 2002.
8. T. Fahle and M. Sellmann. Constraint Programming Based Column Generation with Knapsack Subproblems. *Annals of Operations Research*, to appear.
9. F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proc. of CP'00*, Springer LNCS 1894, pp. 187–201, 2000.
10. M.C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. *Academic Press, New York*, 1991.
11. M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees *Operations Research* 18:1138–1162, 1970.
12. M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part II *Mathematical Programming* 1:6–25, 1971.
13. J.Y. Hsiao, C.Y. Tang, R.S. Chang. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Information Processing Letters*, 43(5):229–235, 1992.
14. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint Programming based Column Generation. *Proceedings of the CP'99*, Springer LNCS 1713: 261–274, 1999.

15. ILOG. ILOG SOLVER. Reference manual and user manual. V5.0, ILOG, 2000.
16. M. Lehradt. Basisalgorithmen für ein TV Anytime System, *Diploma Thesis*, University of Paderborn, 2000.
17. S. Martello and P. Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1:169–175, 1977.
18. M. Milano. *Integration of Mathematical Programming and Constraint Programming for Combinatorial Optimization Problems*, Tutorial at CP2000, Sept. 2000, Singapore.
19. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley Interscience, 1988.
20. M. Sellmann and T.Fahle. Coupling Variable Fixing Algorithms for the Automatic Recording Problem. *9th Annual European Symposium on Algorithms (ESA 2001)*, Springer LNCS 2161: 134–145, 2001.