Alcom-FT Technical Report Series
ALCOMFT-TR-01-87

Coupling Variable Fixing Algorithms 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

Variable fixing is an important technique when solving combinatorial optimization problems. Unique profitable variable values are detected with respect to the objective function and to the constraint structure of the problem. Relying on that specific structure, effective variable fixing algorithms (VFAs) are only suited for the problems they have been designed for. Frequently, new combinatorial optimization problems evolve as a combination of simpler structured problems. For such combinations, we show how VFAs for linear optimization problems can be coupled via Lagrangian relaxation. The method is applied on a multimedia problem incorporating a knapsack and a maximum weighted stable set problem.

1 Introduction

Reduction algorithms are of great importance when combinatorial optimization problems have to be solved exactly. The tightening of problem formulations within a branch-and-bound approach improves on the quality of the bounds computed as well as on the approach's robustness. Given a maximization problem $P(x)$ where $x \in \{0,1\}^n$, $n \in \mathbb{N}$, the idea of variable fixing is to use upper bound information to detect unique profitable assignments for a variable: If an upper bound on $P(x_{|x_i=k})$, $k \in \{0,1\}$, drops below the best known solution value, then we can set $x_i \leftarrow 1-k$.

Frequently, constraints of optimization problems can be grouped such that the overall problem can be viewed as a combination of two or more simpler structured problems. Assuming that efficient variable fixing algorithms (VFAs) for these subproblems exist, their independent application usually does not yield an effective algorithm to perform variable fixing for the combined problem. The reason for this is that tight bounds on the objective cannot be obtained by taking only a subset of the constraints into account.

For a multimedia application incorporating a knapsack problem (KP) and an interval graph maximum weighted stable set problem, we show exemplary how two VFAs for linear optimization problems can be coupled via Lagrangian relaxation to achieve an effective reduction algorithm for the combined problem.

The paper is structured as follows: In Section 2, we introduce the Automatic Recording Problem, that can be viewed as a combination of a knapsack problem and a maximum weighted stable set problem on interval graphs. In Section 3, we introduce an efficient VFA for the latter problem. Then, in Section 4, we show how it can be coupled with a previously developed VFA for KP via Lagrangian relaxation. Finally, in Section 5 we give numerical results.

2 The Automatic Recording Problem

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

*This work was partly supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

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 (see [2]). The 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. The digital video recorder is supposed to record movies 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 (10h of MPEG-2 video need about 18 GB). And second, only one video can be recorded at a time.

More formally, we define the problem as follows:

Definition 1 *Let $n \in \mathbb{N}$, $V = \{0, \ldots, n-1\}$ the set of movies, $start(i) < end(i)\ \forall\ i \in V$ 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 movie $i \in V$, and call two movies $i, j \in V$* 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 movie 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).

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

2.1 A Mathematical Programming Formulation

We focus on exact approaches to solve the ARP. Using mathematical programming, the problem can be stated as an integer program (IP):

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

The objective function maximizes the user satisfaction. Constraints of the form $x_i + x_j \le 1$ ensure that for overlapping intervals I_i, I_j at most one movie 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.*

On interval graphs, the computation of maximal cliques can be performed in time $\Theta(n \log n)$. Then, restrictions of the form $\sum_{i \in C_p} x_i \le 1\ \forall\ 0 \le p < m$ imply that $x_i + x_j \le 1$ for all nodes $i, j \in V$ whose corresponding intervals overlap. On the other hand, if $x_i + x_j \le 1$ for all overlapping intervals, it is also true that $\sum_{i \in C_p} x_i \le 1\ \forall\ 0 \le p < m$. Thus, IP (1) is equivalent to

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

Although there exist methods that do not split the search space – like e.g. cutting plane algorithms – to solve a (mixed) integer program, branch-and-bound approaches have proofed to be efficient, widely applicable and thus are most commonly used. In every search node, 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. Fixing variables can help to improve on the performance of a branch-and-bound search if the VFA is both, effective and efficient. Effective means, that the algorithm must have an impact, i.e., it has to be able to fix many variables, whereas the efficiency measures how quickly the routine works.

The effectivity of a VFA 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 fixing variables effectively, that is, other than by probing via full reoptimization, which is inefficient. On the other hand, the fixing of variables with respect to reduced cost information can be done quickly, but is not very effective.

The ARP can be viewed as a combination of two simpler optimization problems: a knapsack problem, and a maximum weighted stable set problem on an interval graph. For the knapsack problem, an efficient VFAs exists [4]. It runs in time $\Theta(n \log n)$, and in amortized linear time for $\Omega(\log n)$ search nodes. In the following, we develop a VFA for the maximum weighted stable set substructure (MWSSP) of the ARP.

3 Maximum Weighted Stable Set on Interval Graphs

On interval graphs, the problem of finding a maximum weighted stable set [6] can be solved easily in time $\Theta(n \log n)$ [7]. However, the existing algorithms based on sweep line or dynamic programming approaches neither provide dual values for the maximal clique constraints (which is important for the coupling of VFAs as we shall see later), nor do they suggest how variable fixing could be performed efficiently.

3.1 A Mathematical Programming Approach

We present an algorithm based on mathematical programming for the for the maximum weighted stable set problem (MWSSP) on interval graphs that provides us with dual information as a by product, and that can be extended to an efficient VFA for the problem.

Remark 3 *$\bigcup_{0\le p<m} C_p = V$, because $\{i\}$ is a conflict clique $\forall\ i \in V$. Thus, there exists a maximal conflict clique C_p, $0 \le p < m$, such that $i \in C_p$.*

Definition 4 *We set $max_start : M \to \mathbb{N}$, $max_start(C) := \max_{i\in C}\{start(i)\}$.*

Lemma 5 *The function max_start is injective.*

Without loss of generality, we may thus assume that the conflict cliques are ordered with respect to max_start, i.e. $max_start(C_p) < max_start(C_q)\ \forall\ 0 \le p < q < m$.

Lemma 6 *Let $0 \le p < r < m$ and $i \in C_p \cap C_r$. Then, $i \in C_q\ \forall\ p < q < r$.*

Corollary 7 *$m \le n$.*

Definition 8 *We set* $R_p := C_p \setminus C_{p+1}\ \forall\ 0 \leq p < m-1$ *and* $R_{m-1} := C_{m-1}$*, and call every such* R_p *a* (max_start) rest clique.

Remark 9 *The rest cliques form a partition of* V.

Let $C_0, \ldots, C_{m-1}$ denote the maximal conflict cliques ordered according to max_start, and consider IP (3) that evolves from IP (2) by dropping the capacity restriction.

The maximal conflict clique restrictions 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 I_i and I_j, it is also true that $\sum_{i \in C_p} x_i \leq 1\ \forall\ 0 \leq p < m$. Thus, the IP (3) solves the MWSSP on interval graphs.

In the following, with $A \in \{0,1\}^{m \times n}$ we denote the corresponding matrix to IP (3), i.e. $A = (a_{pi})_{0 \leq p < m, 0 \leq i < n}$ with $a_{pi} = 1$ iff $i \in C_p$.

Theorem 10 *The corresponding matrix* A *of IP (3) is an interval matrix.*

Corollary 11 *IP (3) is totally unimodular.*

Let $c_i := -p_i$ for all $0 \leq i < n$. With Corollary 11 it is now possible to solve the MWSSP on interval graphs as a Linear Program:

$$\begin{array}{lrcll} \text{Minimize} & \sum_{0 \leq i < n} c_i x_i & & & (LP\ 4) \\ \text{subject to} & \sum_{i \in C_p} x_i & \leq & 1 \quad \forall\ 0 \leq p < m & \\ & x & \geq & 0 & \end{array}$$

Notice, that with Remark 3 the maximal conflict clique restrictions imply that $x \leq 1$.

A Pivot Selection Strategy We use the Simplex method for solving LP (4). Let $R_0, \ldots, R_{m-1}$ denote the (max_start) rest cliques. In iteration $0 \leq t < m$, we choose $q := t$ as pivot row and $j \in R_q$ with the highest reduced costs as pivot column. If the reduced costs of j are lower than 0, we perform a pivot step. Otherwise we proceed with the next iteration immediately.

Theorem 12 *After* m *such iterations, the Simplex tableau is primal and dual feasible.*

Before we can proof the above Theorem 12, we need to state two more Lemmata first. In the following, we refer to the matrix $A^t \in \{-1,0,1\}^{m \times n}$ after $0 \leq t \leq m$ Simplex iterations with $(a^t_{pi})_{0 \leq p < m, 0 \leq i < n}$, to the right hand side $b^t \in \mathbb{R}^m$ with $(b^t_p)_{0 \leq p < m}$, and to the reduced costs $\hat{c}^t$ with $(\hat{c}^t_i)_{0 \leq i < n}$.

We first proof that our pivot selection preserves primal feasibility. We observe that $x = 0$ is primal feasible as $b^0_p = 1 \geq 0,\ \forall\ 0 \leq p < m$. To assure the maintenance of primal feasibility, we must show that $b^t \geq 0,\ \forall\ 0 \leq t \leq m$. To do so, we proof the following

Lemma 13 *Let* $0 \leq p < m$, $0 \leq t \leq m$, $0 \leq i < n$. *Then,*

(a) $p \geq t-1$ *implies that* $a^t_{pi} = a^0_{pi}$ *and* $b^t_p = b^0_p = 1$

(b) $b^t_p = 0$, $i \in \bigcup_{p \geq t} R_p$ *implies* $a^t_{pi} \in \{-1, 0\}$

(c) $b^t_p = 1$, $i \in \bigcup_{p \geq t} R_p$ *implies* $a^t_{pi} \in \{0, 1\}$

(d) $b^t_p \in \{0, 1\}$

Proof We induce over t:
$\boldsymbol{t = 0}$ **:** $b^0_p = 1$ and $a^0_{pi} \in \{0,1\}\ \forall\ 0 \leq p < m$ and $0 \leq i < n$.
$\boldsymbol{t \to t+1}$ **:** Let $t < m$ and $j \in R_t$ the pivot column in step t. If the reduced costs of column j are greater or equal zero, then we are done since $b^{t+1} = b^t$ and $A^{t+1} = A^t$. Otherwise, we set $q := t$ and choose a^t_{qj} as pivot element. By induction hypothesis (a) we know that $b^t_q = b^0_q = 1$, and that $a^t_{qj} = a^0_{qj}$. Now, because of $j \in R_q \subseteq C_q$, $a^0_{qj} = 1$. Thus, our pivot element is equal to 1.
(a) Therefore,

$$a_{qi}^{t+1} = a_{qi}^t = a_{qi}^0, \qquad \text{and} \qquad b_q^{t+1} = b_q^t = b_q^0 = 1 \qquad \forall\, 0 \le i < n$$

Now let $t < p < m$. According to our pivot selection strategy, $j \in R_q = C_q \setminus C_{q+1} \Rightarrow j \notin C_{q+1}$. This, and the interval property of the matrix A imply $a_{pj}^t = a_{pj}^0 = 0$ for $p = t+1$, and thus for all $p > t$. Thus, in step t none of the rows $p > t$ changes, i.e.

$$a_{pj}^{t+1} = a_{pj}^t = a_{pj}^0 \qquad \text{and} \qquad b_p^{t+1} = b_p^t = b_p^0.$$

For $p \ge t$, (a) implies (b)–(d). In the following we thus assume $p < t$. As for all $p < t$ with $a_{pj}^t = 0$ row p does not change, we only need to consider $p < t$ with $a_{pj}^t \neq 0$. And as A is totally unimodular, $a_{pj}^t \in \{-1, 1\}$.

Finally, let $i \in \bigcup_{p \ge t} R_p$ with $a_{qi}^t = a_{qi}^0 = 0$. Due to the interval matrix property of A and $a_{pi}^0 = 1$ for some $p > t = q$, we know that then already $a_{pi}^0 = 0\ \forall\ p \le q$. Moreover, as all pivot elements until step t were chosen from rows $p < q$, it holds that $a_{pi}^t = a_{pi}^0 \in \{0, 1\}$, and $b_p^t = b_p^0 = 1$ for all $q < p < m$. Therefore, we only need to consider $a_{qi}^t = a_{qi}^0 = 1$.

(b)–(d) Let $p < t$, $i \in \bigcup_{p \ge t} R_p$, $a_{qi}^t = 1$ and $a_{pj}^t \in \{-1, 1\}$. Using induction hypothesis (d), we know that $b_p^t \in \{0, 1\}$.

First, assume that $b_p^t = 0$. By induction hypothesis (b) we know, that $a_{pj}^t = -1$ and $a_{pi}^t \in \{-1, 0\}$. Thus,

$$b_p^{t+1} = b_p^t + 1 = 1 \in \{0, 1\} \qquad \text{and} \qquad a_{pi}^{t+1} = a_{pi}^t + 1 \in \{0, 1\}.$$

Now let us assume $b_p^t = 1$. Then, by induction hypothesis (c), we know that $a_{pj}^t = 1$, and $a_{pi}^t \in \{0, 1\}$. Thus,

$$b_p^{t+1} = b_p^t - 1 = 0 \in \{0, 1\} \qquad \text{and} \qquad a_{pi}^{t+1} = a_{pi}^t - 1 \in \{-1, 0\}.$$

□

Now, we show that after at most m iterations we achieve dual feasibility.

Lemma 14 *Let $1 \le t \le m$. Then,*

(a) $\hat{c}_i^t \ge 0$ for all $i \in R_{t-1}$.

(b) $i \in \bigcup_{0 \le p < t < m} R_p$ implies $\hat{c}_i^{t+1} = \hat{c}_i^t$.

Proof (a) Let $j \in R_{t-1}$ denote the pivot column in step $t-1$. If $\hat{c}_j^{t-1} \ge 0$ we are done, as then $\hat{c}_j^t = \hat{c}_j^{t-1} \ge \hat{c}_i^{t-1} \ge 0$ for all $i \in R_{t-1}$. We may therefore assume $\hat{c}_j^{t-1} < 0$. In Lemma 13, we have already shown that the pivot element is $a_{t-1,j}^{t-1} = 1$, and $a_{t-1,i}^{t-1} = a_{t-1,i}^0\ \forall\ 0 \le i < n$. In particular, we know that $a_{t-1,i}^{t-1} = a_{t-1,i}^0 = 1\ \forall\ i \in R_{t-1} \subseteq C_{t-1}$. We conclude that $\hat{c}_i^t = \hat{c}_i^{t-1} - \hat{c}_j^{t-1} \ge 0$. (b) Let $0 \le p < t < m$, $i \in R_p$ and $t \le q < m$. Lemma 13 states that $a_{qi}^t = a_{qi}^0$. More precisely, due to the interval matrix property of A, and $i \in R_p = C_p \setminus C_{p+1}$, we know that $a_{ti}^t = a_{ti}^0 = 0$. Therefore, $\hat{c}_i^{t+1} = \hat{c}_i^t$. □

Corollary 15 *After m iterations we achieve dual feasibility.*

Proof Let $0 \le i < n$. Then there exists $1 \le t \le m$ such that $i \in R_{t-1}$. It holds: $\hat{c}_i^m = \hat{c}_i^{m-1} = \ldots = \hat{c}_i^t \ge 0$. □

Proof of Theorem 12 In Lemma 13 and Corollary 15, we have shown that after $m \le n$ iterations the Simplex tableau is primal and dual feasible. □

We have shown how the MWSSP on interval graphs can be stated as a totally unimodular LP. Further, we have proven a feasible pivot selection strategy that yields an optimal tableau after at most n Simplex iterations.

An Efficient Simplex Realization In the following, we develop an efficient $\Theta(n \log n)$-time algorithm to calculate a set $Q \subseteq \{0, \ldots, n-1\}$ with $I_i \cap I_j = \emptyset \ \forall \ i, j \in Q$, such that $c_Q = \sum_{i \in Q} c_i$ is minimal. Most importantly, the algorithm also provides us with dual information as a byproduct. To establish that algorithm, we show how the Simplex calculations according to the pivot strategy developed can be performed efficiently.

Theorem 16 *Let $0 \leq l < m$, and let $(j_0, \ldots, j_l) \in \{0, \ldots, n-1\}^l$ denote the sequence of pivot columns according to Section 3.1 for which a pivot step has been performed. Further, for all $0 \leq k \leq l$ let $0 \leq p_k < m$ with $j_k \in R_{p_k}$. Set $Q := \{j_k \mid 0 \leq k \leq l$ and $I_{j_k} \cap I_{j_r} = \emptyset \ \forall \ k < r \leq l\}$. Then c_Q is minimal.*

Proof We induce over m:
$\boldsymbol{m = 1}$ **:** As there is only one row, at most one pivot step is being carried out. If no pivoting is taking place, the initial tableau is optimal ($l = 0$), and $Q = \emptyset$ is an optimal solution. Otherwise, $p_0 = 0$, and we only have to show that $b_0^1 = 1$. But according to Lemma 13, $b_0^1 = b_0^0 = 1$.
$\boldsymbol{m \to m+1}$ **:** Now assume that there are $m+1$ maximal conflict cliques. Again by applying Lemma 13, we know that $b_{p_l}^{m+1} = b_{p_l}^{p_l} = b_{p_l}^0 = 1$. Therefore, there exists an optimal solution

$$Q \subseteq \{j_r \mid 0 \leq r \leq l\}, \quad \text{with} \quad j_l \in Q.$$

Let $k := \min\{r \mid 0 \leq r < l, \ j_l \in C_r\}$. By setting $N := \bigcup_{k \leq r < l} C_r$, we know that $N \cap Q = \emptyset$. Thus, there exists an optimal solution $Q = \{j_l\} \cup S$, where

$$S \subseteq V \setminus N = \bigcup_{0 \leq r < k} R_r, \quad \text{and it holds} \quad I_i \cap I_j = \emptyset \qquad \forall \ i, j \in S.$$

As $c_Q = c_{j_l} + c_S$, we can construct an optimal solution by finding such a set S with a minimal value c_S. If $k = 0$, it holds $S = \emptyset$, and we are done. Otherwise, we have to solve an MWSSP with $k \leq m$ maximal conflict cliques to find such a set S. By setting up the according LP, we find that the sequence of pivot elements is exactly $(j_0, \ldots, j_{k-1})$. We apply our induction hypothesis and achieve

$$S = \{j_r \mid 0 \leq r \leq k-1 \text{ and } I_{j_r} \cap I_{j_s} = \emptyset \ \forall \ r < s \leq k-1\}.$$

Thus,

$$Q = \{j_l\} \cup S = \{j_r \mid 0 \leq r \leq l \text{ and } I_{j_r} \cap I_{j_s} = \emptyset \ \forall \ r < s \leq l\}$$

is an optimal solution to the MWSSP on interval graphs. □

The above Theorem 16 allows us to construct an optimal solution if we know the sequence of pivot elements. But according to Section 3.1, calculating the sequence of pivot elements is an easy task, if only we can determine the reduced costs quickly:

Lemma 17 *Let $0 \leq t < m$, $(j_0, \ldots, j_{m-1}) \in V^m$ be the sequence of pivot columns according to our pivot selection strategy, and let $d : \{0, \ldots, m-1\} \to \{0, 1\}$ with $d(t) := 1 \Leftrightarrow \hat{c}_{j_t}^t < 0$. Further, we set $q_t := t$, $0 \leq q_t < m$, and let $j_t \in R_{q_t}$.*

Let $z^t \in \mathbb{R}$ denote the objective function value and $a_{q_t j_t}^t$ be the pivot element in iteration t. Further, let $f_i := \min\{p \mid 0 \leq p < m, \ i \in C_p\} \ \forall \ 0 \leq i < n$ denote the index of the first maximal conflict clique that node i belongs to, i.e. $a_{f_i i}^0 = 1$, and $a_{pi}^0 = 0 \ \forall \ 0 \leq p < f_i$. Finally, let $g_i^t := z^{f_i} - z^t$ if $f_i \leq t$, and otherwise $g_i^t := 0 \ \forall \ 0 \leq i < n$. Then,

(a) $z^{t+1} = z^t + \hat{c}_{j_t}^t \cdot d(t) \leq z^t$.

(b) $z^t = \sum_{0 \leq r < t} \hat{c}_{j_r}^r \cdot d(r)$.

(c) $\hat{c}_i^t = c_i + g_i^t \qquad \forall \ i \in \bigcup_{t \leq p < m} R_p$

Proof **(a)** If no pivoting takes place, $\hat{c}^t_{j_t} \geq 0 \Rightarrow d(t) = 0$, and $z^{t+1} = z^t$. Otherwise, $d(t) = 1$. According to Lemma 13, $a^t_{q_t j_t} = a^0_{q_t j_t} = 1$, and $b^t_{q_t} = b^0_{q_t} = 1$. Thus, $z^{t+1} = z^t + \hat{c}^t_{j_t} \cdot b^t_{q_t} / a^t_{q_t j_t} = z^t + \hat{c}^t_{j_t} < z^t$.
(b) is a simple implication of (a).
(c) Let $i \in \bigcup_{t \leq p < m} R_p$. First, assume that $f_i > t$. Then, $a^0_{pi} = 0 \; \forall \; 0 \leq p \leq t$. As all pivot elements until step t were chosen from rows $0 \leq p \leq t$, we have that $\hat{c}^t_i = c_i = c_i + g^t_i$.

So let $f_i \leq t$. Using (b), we see that $g^t_i = z^{f_i} - z^t = -\sum_{f_i \leq r < t} \hat{c}^r_{j_r} \cdot d(r)$. As $a^0_{pi} = 0 \; \forall \; 0 \leq p < f_i$, we know that $\hat{c}^{f_i}_i = c_i$. Let $f_i \leq r < t$. With Lemma 13, it holds $a^r_{q_r j_r} = a^0_{q_r j_r} = 1$, and also $a^r_{q_r i} = a^0_{q_r i} = 1$. Thus, $\hat{c}^{r+1}_i = \hat{c}^r_i - \hat{c}^r_{j_r} \cdot d(r)$, and hence $\hat{c}^t_i = \hat{c}^{f_i}_i - \sum_{f_i \leq r < t} \hat{c}^r_{j_r} \cdot d(r) = c_i + g^t_i$. □

An Algorithm providing Dual Information With Theorem 16 and Lemma 17, we can formulate an efficient algorithm solving the MWSSP on interval graphs that provides us with dual values as a byproduct. In phase 1, we determine the (max_start) rest cliques R_p, with $0 \leq p < m$, and the according values $f_i \; \forall \; 0 \leq i < n$. This can be done in time $\Theta(n \log n)$.

Phase 2 consists of m iterations: First, we set $z^0 := 0$. In each iteration $0 \leq t < m$ we calculate $\hat{c}^t_i = c_i + z^{f_i} - z^t \; \forall \; i \in R_t$, and $j_t \in R_t$ with $\hat{c}^t_{j_t} = \min_{i \in R_t}\{\hat{c}^t_i\}$. If $\hat{c}^t_{j_t} \geq 0$, we set $z^{t+1} := z^t$, otherwise $z^{t+1} := z^t + \hat{c}^t_{j_t}$. Finally, we set $t := t + 1$ and proceed with the next iteration.

With Remark 9, we know that the sets R_p form a partition of V. Thus, all nodes $0 \leq i < n$ are being looked at exactly once to calculate the reduced costs. Also, in all computations of the pivot columns, each node is incorporated only once. Therefore, phase 2 takes time $\Theta(n)$.

After m iterations, we know the shortest path value z^m, as well as the sequences $(j_0, \ldots, j_l) \in V^l$ and $(p_0, \ldots, p_l) \in \{0, \ldots, m-1\}^l$, $0 \leq l < m$, of pivot columns and rows for which a pivot step has been performed. By applying Theorem 16, we can construct a shortest path out of this information in linear time. Moreover, as the rest cliques of the underlying interval graph are independent of the objective function, we achieve an incremental linear time algorithm for $\Omega(\log n)$ calls with different objectives.

Most importantly, we get dual values as a byproduct. By looking at the optimal tableau, we find that the optimal dual variable for each maximal clique constraint $0 \leq t < m$ for which a pivot step has been carried out has value $-\hat{c}^m_{j_t}$. All other dual values are 0.

3.2 Variable Fixing

Now, we want to use the previously developed algorithm to perform variable fixing. To develop a VFA, we reinterpret the problem as finding a shortest path in a node-weighted co-interval graph. We introduce an artificial source σ and sink τ before and after all other nodes, and define $Path_{\sigma,\tau}$ as the set of paths from σ to τ. The value c_P of a path $P \in Path_{\sigma,\tau}$ is defined as $c_P := \sum_{i \in P \setminus \{\sigma,\tau\}} c_i$.

Definition 18 *Given an upper bound $Z \in \mathbb{R}$, we define $Rem(Z)$ and $Req(Z)$ by*

$$Rem(Z) := \{0 \leq i < n \mid \forall \; P \in Path_{\sigma,\tau}, i \in P : \; c_P \geq Z\}$$

$$Req(Z) := \{0 \leq i < n \mid \forall \; P \in Path_{\sigma,\tau}, i \notin P : \; c_P \geq Z\}.$$

The variable fixing problem based on shortest path information (VFSPP) then is to determine $Rem(Z)$ and $Req(Z)$.

Removing Nodes To compute $Rem(Z)$, we need to find out the value of the shortest path from the source σ via node i to the sink τ for all $i \in \{0, \ldots, n-1\}$. This can be done by calculating the shortest path distances from the source and to the sink. The algorithm in Section 3.1 can easily be adapted to determine the shortest path distances from the source to each node $j \in \{0, \ldots, n-1\}$. For $0 \leq i < n$, let $f_i := \min\{p \mid 0 \leq p < m, \; i \in C_p\}$ denote the index of the first maximal conflict clique that node i belongs to. Then, we have to solve the following Linear Program

$$\begin{array}{lrcll} \text{Minimize} & \sum_{0 \leq i < j} c_i x_i & & & (LP\,5) \\ \text{subject to} & \sum_{i \in C_p} x_i & \leq & 1 \quad \forall \; 0 \leq p < f_j & \\ & x & \geq & 0 & \end{array}$$

According to the previously developed theory, the minimal objective for the above LP (5) is exactly z^{f_j}. And thus, the shortest path distance of node j is exactly $d_j = z^{f_j} + c_j$.

A similar theory as presented before shows that the shortest path distances to the sink can be determined by applying the algorithm of Section 3.1 using the last clique belongings

$$l_i := \max\{p \mid 0 \leq p < m,\ i \in C_p\} \qquad \forall\ 0 \leq i < n,$$

and the min_end rest cliques, where

$$min_end : M \to \mathbb{N}, min_end(C) := \min_{i \in C}\{end(i)\}.$$

Solving LP (4) in this inverse manner yields objective function values ${}^{\tau}z^t$ for all iterations $0 \leq t \leq m$.

Then, the shortest path distance to the sink is ${}^{\tau}d_j = {}^{\tau}z^{m-l_j-1} + c_j$. With those values at hand, we can determine the shortest path value e_j through node $0 \leq j < n$ by $e_j = d_j + {}^{\tau}d_j - c_j$. Then,

$$Rem(Z) = \{0 \leq i < n \mid e_i \geq Z\}.$$

We conclude that $Rem(Z)$ can be computed in time $\Theta(n \log n)$ and in amortized linear time for $\Omega(\log n)$ calls of the VFA.

Now we know how to determine nodes that should be removed from the graph. Of course, other constraints may as well remove nodes. In both cases, we must be able to take these changes into consideration efficiently for the next call to our routine. Without going into implementation details, we note that the data structures storing the rest cliques as well as the first and last clique belongings can be compressed in linear time to delete any number of nodes from the graph.

Requiring Nodes To compute $Req(Z)$, we need to identify all nodes that must be an element of any path having a value lower than Z. Obviously, only nodes on the shortest path S potentially have this property. For every node $j \in S$, we thus need to find out what the value of a shortest path P with $j \notin P$ is.

Remark 19 *Let $0 \leq f_j \leq l_j < m$ denote the first and the last clique belongings of j. Further, let P be the shortest path with $j \notin P$. Obviously, it either holds that $I_j \cap I_i = \emptyset\ \forall\ i \in P$, or $I_j \cap I_i \neq \emptyset\ \exists\ i \in P$. In the first case, we know that the value of the shortest path not using the time interval I_j has the value*

$$c_P = z^{f_j} + {}^{\tau}z^{l_j} - 2c_j.$$

In the second case, we do not know how to determine the value of P efficiently. However, after we have determined and deleted $Rem(Z)$ from the graph, we only have to check if there exists any node $i \in \{0, \ldots, n-1\}$, $i \neq j$, with $I_j \cap I_i \neq \emptyset$. Because if such a node i exists, there also exists a path $\bar{P}$ with $i \in \bar{P}$ and $c_{\bar{P}} < Z$ (otherwise i would have been deleted before). As i and j overlap, we further know that $j \notin \bar{P}$. Thus, in $\bar{P}$ we have found a path not covering j with a value lower than Z. Therefore, $j \notin Req(Z)$. On the other hand, if no such node i exists, the second case is obsolete, and we only need to consider the first case.

By making that observation, we can determine $Req(Z)$ in linear time: First, for all $j \in P$ we check whether there exists a node in the shrunk graph that overlaps with j. Without specifying the implementation details here, we just note that this can be done in constant time for each node j. If no overlapping node exists, we compute $z^{f_j} + {}^{\tau}z^{l_j} - 2c_j$ and check whether this value is lower than Z. If not, we add j to $Req(Z)$, otherwise we do not.

Now, we have an efficient algorithm at hand to compute $Req(Z)$. Obviously, other constraints and branching decisions must be taken into account when our procedure is being called next. Thus, we have to be able to transform our graph in such a way that from now on every path covers the new required nodes. At a first glance this sounds problematic, as an easy approach would delete all arcs going around the required nodes. This procedure would cause the resulting graph not to have the co-interval property anymore.

We can force the shortest path to visit the required nodes by making them extremely cheap: Let $Req \subseteq \{0, \ldots, n-1\}$ the set of (currently) required nodes. Further, let $M \gg 0$ sufficiently large[1]. Then, we set

[1] Assuming that $\min_{i \in V}\{c_i\} < 0$, a valid setting for M is for example $M := n \cdot (1 + \max_{i \in V}\{c_i\} - \min_{i \in V}\{c_i\})$

$\bar{c}_j := c_j - M \ \forall \ j \in Req$, and $\bar{c}_j := c_j \ \forall \ j \notin Req$. We use $\bar{c}$ instead of c as our objective and check whether the shortest path value is lower than $Z - |Req| \cdot M$. If not, either two required nodes overlap, or the shortest path value in the original graph exceeds Z. Moreover, by determining $Rem(Z - |Req| \cdot M)$, we find all nodes that overlap with some required node plus all nodes that would cause the shortest path in the original graph to exceed the threshold Z. We conclude:

Theorem 20 *The VFSPP can be solved in time $\Theta(n \log n)$ or in amortized linear time for $\Omega(\log n)$ incremental calls of the variable fixing algorithm.*

4 Coupling Variable Fixing Algorithms via Lagrangian Relaxation

An obvious approach to solve the ARP exactly is to apply a branch-and-bound algorithm using linear relaxation bounds for pruning and the existing VFAs for knapsack and maximum weighted stable set problems on interval graphs for the fixing of variables.

Although the existing VFAs are efficient and effective for the substructures they have been designed for, their application for the ARP is not. This is because neither assuming an unlimited storage capacity nor ignoring the fact that only one movie can be recorded at a time yields a tight bound on the objective for the combined problem. But the accuracy of the upper bound is essential for the effectiveness of a VFA. An accurate bound can only be computed by looking at the entire problem, i.e., it cannot be achieved by looking at either one constraint family only.

The linear relaxation bound can easily be obtained by applying a standard LP solver or by using specialized methods tailored for this specific application (see section 5.1). That bound 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 the fixing of variables. Applying conventional reduced cost variable fixing techniques only indirectly exploits the structure of the problem and is therefore not effective enough, whereas to perform probing via full reoptimization is very costly and inefficient.

Lagrangian relaxation (see e.g. [1] for an introduction) allows us to bring together the advantages of a tight continuous global bound and the existing VFAs that exploit the special structure of their respective constraint families. As the stable set VFA 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

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

The Lagrange multiplier problem then is to solve

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

For every $\lambda \geq 0$, $z(\lambda)$ is a valid upper bound on the objective. Therefore, we can apply the VFA for maximum weighted stable set problems on interval graphs each time we solve the Lagrangian subproblem. After we have found an optimal Lagrange multiplier λ^*, i.e. $z(\lambda^*) \leq z(\lambda) \ \forall \ \lambda \geq 0$, we can use the (optimal) 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

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

Relaxations of this problem again yield an upper bound on the objective, and we can apply the knapsack VFA from [4].

In general, two linear optimization constraint families for which efficient VFAs are known can be combined effectively by computing Lagrangian multipliers for the first, using the second for fixing variables in each Lagrangian subproblem $L(\lambda)$, and then handing back dual information of the optimal $L(\lambda^*)$ to fix variables with respect to the first constraint family with the corresponding (optimal) reduced cost objective. This procedure even strengthens the bound on the objective, as variable fixing is also done during the bound computation. However, if variables are being fixed 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.

5 Numerical Results

5.1 Implementation Details

As we mainly focus on the general principle of how existing VFAs can be exploited more effectively by tying them together using Lagrangian relaxation, so far we have left out 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 variable fixing (referred to as *no fixing (F-0)*). The second uses the VFAs for knapsack and maximum weighted stable set problems on the original objective (*fixing 1 (F-1)*). The third and the fourth approach (*fixing 2 (F-2)* and *fixing 3 (F-3)*) realize the idea of coupling VFAs for linear optimization problems via Lagrangian relaxation. *F-2* calls for the fixing of variables just once after the Lagrangian dual has been solved, whereas *F-3* also performs maximum weighted stable set variable fixing during the search for optimal Lagrange multipliers.

Continuous Bound Computation For pruning, the computation of a linear bound on the objective is needed. *F-2* and *F-3* obviously use the objective value corresponding to $L(\lambda^*)$ 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 λ^* 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 $\Omega(\log n)$ search nodes. Finally, in every search node we add $O(n \log n)$ for the succeeding knapsack VFA. Therefore, the coupled VFA 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[2] with maximal efficiency p_i/w_i as branching variable.

5.2 Experiments

All experiments were performed on a PC with an AMD-Athlon 600 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 [8].

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

Test set			*runtime in sec. (choice points)*			
min.	*#ch*	*Ømov.*	*no fixing F-0*	*fixing F-1*	*fixing F-2*	*fixing F-3*
360	5	29.0	0.1 (27.2)	0.1 (15.6)	0.0 (14.6)	0.0 (10.9)
360	20	115.6	0.4 (49.9)	0.2 (32.3)	0.1 (24.2)	0.2 (17.3)
360	50	285.2	1.1 (52.5)	0.8 (37.3)	0.3 (25.7)	0.5 (22.1)
360	100	569.0	2.9 (96.7)	2.1 (67.9)	0.8 (36.7)	1.2 (30.0)
720	5	52.4	0.1 (69.3)	0.1 (33.3)	0.1 (26.0)	0.1 (17.6)
720	20	208.3	0.6 (91.7)	0.4 (69.0)	0.3 (36.5)	0.4 (29.4)
720	50	523.4	4.2 (305.5)	2.4 (166.6)	1.3 (58.7)	2.0 (52.3)
720	100	1048.3	18.8 (680.8)	9.1 (326.3)	5.1 (108.6)	7.6 (95.5)
1440	5	101.4	0.8 (130.4)	0.7 (98.2)	0.2 (48.0)	0.4 (38.5)
1440	20	402.9	7.9 (479.6)	5.3 (306.1)	1.6 (103.0)	2.5 (81.9)
1440	50	1013.4	37.8 (1461.1)	19.5 (734.4)	11.1 (187.9)	13.8 (178.5)
1440	100	2019.8	291.0 (5351.2)	108.2 (2028.1)	55.6 (530.2)	71.0 (439.9)
4320	5	291.7	4.7 (508.2)	6.0 (327.3)	2.0 (138.9)	3.1 (121.6)
4320	20	1175.0	177.4 (3003.1)	111.7 (1897.2)	36.6 (468.4)	42.9 (401.2)
4320	50	2949.7	12227.8 (149877.2)	5647.5 (67864.5)	1191.5 (4744.5)	1197.3 (4703.5)
7200	5	487.8	64.0 (3272.2)	43.5 (2705.4)	17.5 (878.0)	17.9 (434.5)
7200	20	1956.7	5210.3 (60839.2)	1734.4 (30676.4)	455.9 (3433.9)	490.1 (2945.1)

Table 1: Numerical results for the ARP. The first three columns characterize each group of 50 instances by giving the planning horizon in minutes, the number of channels, and the average number of movies. Columns 4 to 7 present running times for the different approaches. In brackets, we give the number of search nodes, in the following referred to as *choice points*.

The ARP test instances were generated by specifying the time horizon and the number of channels. The generator sequentially fills the channels by starting each new movie one minute after the last. First, a class for the next movie is being chosen randomly. That class then determines the intervals from which the length and the profit are chosen randomly (for now, we have only been using 3 different classes of movies). The disc space necessary to store each movie equals its length, and the storage capacity is randomly chosen as 40%–60% of the entire time horizon.

The experiment consists of 50 random instances per test set. For each instance, the approaches *F-0* – *F-3* were run to find and prove an optimal solution. The minutes for the time horizon equal 6 hours, 12 hours, one day, 3 days, and 5 days of digital television, respectively.

It should be noted that 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.

Obviously, the linear continuous bound we use for pruning is rather tight. Except for the 4320-50 and 7200-20 instances, even *F-0* is able to cope with all problem classes fairly well. Thus, it is justified to base pruning and also variable fixing on that bound.

The positive effect of applying a VFA can be seen when comparing *F-0* and *F-1*. Although the latter is not very effective, it is already able to reduce the number of choice points by a factor of $1.5 - 2$. This results in a speed-up of up to 3 for the 7200-20 instances.

F-2 and *F-3* yield a dramatic further reduction of choice points. For the 7200-20 instances, *F-3* only needs 10% of the choice points of *F-1*. For the 4320-50 instances, it even reduces the number of choice points to 7% compared to *F-1*, and to 3% with respect to the number of choice points visited by *F-0*.

The gain in time is not that drastic, as the greater effectivity must be paid for by a higher computational effort per search node. Thus, the maximum speed-up we get between *F-0* and *F-2* is "only" $10 - 11$. And although *F-3* is more effective and outperforms *F-2* regarding the number of choice points, it is slower on all instances. The reason for both is that it performs maximum weighted stable set variable fixing for each Lagrangian subproblem. Of course, if the absolute time spent per choice point was higher (due to other variable fixing algorithms or more costly bound computations for example), the increased effectivity should also result in a faster overall computation. By using a parameter ζ that determines the frequency of maximum weighted stable set variable fixing, we can trade time for effectivity. Like that, we can tune our coupled VFA towards our specific application. To achieve a most effective filtering algorithm, the VFAs

for both substructures could be applied for each Lagrangian subproblem.

A topic of further investigation is the comparison of test sets with an equal number of movies: E.g., instances in 720-100, 1440-50, and 4320-20 all contain about 1000 movies, and the solution time decreases with the number of channels increasing. In a test set with a high number of channels, many nodes are overlapping, hence the maximum weighted stable set VFA can easily eliminate many variables from consideration. With a lower number of channels and a wider time horizon, the knapsack constraint gains more importance. We assume that a vice versa approach (i.e., Lagrange relaxing the non-overlapping constraints and solving a knapsack as subproblem) would be a suitable choice for such a situation.

6 Conclusions and Future Work

For the automatic recording problem, we exemplary introduced the idea of coupling VFAs via Lagrangian relaxation. It allows to combine existing variable fixing routines for linear optimization problems 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 other algorithmic measures such as search nodes. The method is suited for linear optimization problems for which bounds based on continuous or Lagrangian relaxations can be used effectively.

In order to be able to couple VFAs for knapsack and maximum weighted stable set problems, we first had to develop a VFA for the latter. Although for the problem itself algorithms running in time $\Theta(n \log n)$ have been developed before, they did not provide dual information that is needed for the coupling method we introduced. Moreover, to our knowledge no VFA for the MWSSP on interval graphs running in amortized linear time existed before. As the problem occurs as a substructure in many optimization problems, especially in scheduling contexts, the VFA we developed in our view is a contribution that is of general interest and goes beyond the application we presented.

For the multimedia application we introduced, we developed 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 new exact approach presented, a fast and efficient VFA for this type of relaxed non-overlapping constraint is subject to further research.

Acknowledgement We would like to thank Burkhard Monien for many discussions and helpful comments.

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] T. Fahle and M. Sellmann. Constraint Programming Based Column Generation with Knapsack Subproblems. *Proc. of the CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:33–44, 2000.

[5] F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proc. of the CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:45–51, 2000.

[6] M.C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. *Academic Press, New York*, 1991.

[7] 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.

[8] ILOG. ILOG SOLVER. Reference manual and user manual. V5.0, ILOG, 2000.

[9] M. Lehradt. Basisalgorithmen für ein TV Anytime System, *Diploma Thesis*, University of Paderborn, 2000.

[10] 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.

[11] G.L. Nemhauser, L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley Interscience, 1988.

A Appendix – Previously Postponed Proofs

Proof of Lemma 5 Assume $max_start(C_p) = max_start(C_q)$ for some $0 \leq p, q < m$. Then there exist $s_p \in C_p$ and $s_q \in C_q$ such that

$$start(s_p) = max_start(C_p) = max_start(C_q) = start(s_q),$$

and $\forall\ i \in C_p, j \in C_q$:

$$end(i) \geq start(s_p) = start(s_q) \geq start(j)$$

and

$$start(i) \leq start(s_p) = start(s_q) \leq end(j).$$

Thus, all nodes in C_p and C_q are pairwise overlapping. Therefore, $C_p \cup C_q$ is a conflict clique. As C_p and C_q are maximal, we have $C_p = C_p \cup C_q = C_q$. But as $p \neq q$ implies $C_p \neq C_q$, we have $p = q$. □

Proof of Lemma 6 Let $s_p \in C_p, s_q \in C_q, s_r \in C_r$, such that $s_t = max_start(C_t)\ \forall\ t \in \{p, q, r\}$. Further, let $j \in C_q$. Then

$$end(i) \geq start(s_r) > start(s_q) \geq start(j)$$

and

$$start(i) \leq start(s_p) < start(s_q) \leq end(j).$$

Therefore, $C_q \cup \{i\}$ is a conflict clique. As C_q is maximal, $C_q = C_q \cup \{i\}$, i.e. $i \in C_q$. □

Proof of Corollary 7 Let $0 \leq p < m - 1$. $\exists\ i \in C_p$ such that $i \notin C_{p+1}$, as otherwise $C_p \subseteq C_{p+1}$, which contradicts the maximality of C_p or $C_p \neq C_{p+1}$. With Lemma 6, we thus have $i \notin C_q\ \forall\ p < q < m$. Therefore, $|C_q| \leq n - q\ \forall\ 0 \leq q < m$, and thus $m \leq n$. □

Proof of Theorem 10 We have to show that $a_{pi} = a_{ri} = 1$ implies that $a_{qi} = 1\ \forall\ p < q < r$, $0 \leq i < n$. By construction of A, this is equivalent to showing that $i \in C_p \cap C_r$ implies $i \in C_q\ \forall\ p < q < r$. But this is true according to Lemma 6. □

Proof of Corollary 11 Interval matrices are totally unimodular [11]. □