The TV-Break Packing Problem

Thierry Benoist, Eric Bourreau, Benoît Rottembourg

The TV-Break Packing Problem

Thierry Benoist[1], Eric Bourreau[2], Benoît Rottembourg[3]

[1] Bouygues e-lab, 1 av. Eugène Freyssinet, 78061 St Quentin en Yvelines Cedex, France
[2] LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France
[3] TF1, 1 quai Point du Jour, 92100 Boulogne Billancourt, France

tbenoist@bouygues.com, bourreau@lirmm.fr, brottembourg@tf1.fr

Abstract. Instead of selling advertisement spots one by one, some French satellite channels decided in 2002 to modify their commercial offer in order to sell packages of spots. These new General Conditions of Sale lead to an interesting optimization problem that we named the TV-BREAK PACKING PROBLEM (TVBP). We establish its NP-hardness and study various resolutions approaches including linear programming (LP), lagrangian relaxation (LR), constraint programming (CP) and local search (LS). Finally we propose a generic CP/LS hybridization scheme (Branch & Move) whose application to the TVBP obtained the best results in our experiments. Dual upper bounds of the maximal revenue are also computed.

Keywords. Combinatorial Optimization, Constraint Programming, Linear Programming, Local Search.

1. Introduction

Satellite TV channels draw an increasing audience in France (from 1 million to 12 millions viewers in 2002). In addition to subscription fees, a large percentage of their revenues comes from broadcasted advertisements. Since 2001, accurate audience measurements are available for these thematic channels. Share of markets are generally smaller than those of hertzian general interest channels, but target groups (housewives, upper class, 15-24 year old people…) are precisely identified what is an interesting feature from an advertiser point of view since it helps designing sharp marketing strategies.

Instead of selling advertisement "spots" (a slot of 30s during a TV break) one by one, the marketing department of a bundle of satellite channels decided to modify its commercial offer in order to sell *packages* of spots. Several types of packages are proposed, characterized by their *price*, *size* (number of spots) and *shape* (approximate dispatching of spots into keys zones of the week like prime time, week-end, night…). The most innovative aspect of this offer lies in an audience guarantee attached to each type of package, namely a lower bound on the total number of (forecasted) spectators

collected by spots constituting the package. This feature allows advertisers saving time (no need to select each TV-break) without loss of efficiency for the media plan (audience is guaranteed).

For a TV-channel, these packages have to be designed in order to fit the demands of the advertising market. Besides, interaction between these packages must also be taken into account because of the limited capacity of each TV-break: for instance selling too many week-end specific packages can prevent selling packages spanning the whole week. Due to such interactions, packages need to be prepared beforehand. In the above example, building packages on demand may lead to inextricable situations if the first clients all choose packages of the "week-end" family.

The problem considered in this paper takes as input a list of packages (price, size, shape and audience requirement) that the marketing department would like to build for a given week. The objective is to build all these packages, maximizing the total value of those fulfilling their audience constraint. We name it the TV-BREAK PACKING PROBLEM (TVBP). We shall point out in section 2.2 that it can be seen a generalization of the NP-hard 3-PARTITION problem.

The problem is defined in Section 2 and its NP-hardness is proven. Section 3 is devoted to mathematical programming approaches (linear programming and lagrangian relaxation) essentially providing upper bounds of the maximal revenue. In Section 5 a Constraint Programming (CP) model of the problem is designed and a Local Search (LS) improvement process applicable to a solution is introduced. Finally, an original CP/LS hybrid algorithm (*Branch&Move*) is defined and applied to the TVBP, yielding high-quality solutions. After a presentation of computational results we conclude on real-world variants of the TVPB.

2. The TV-Break Packing problem

In this section we give a formal (linear) description of the TV-BREAK PACKING PROBLEM and we establish it NP-hardness. Its underlying flow structure is also emphasized.

2.1. Description

In a TV channel week there is a fixed number of commercial breaks. The TV-BREAK PACKING PROBLEM described below essentially consists in partitioning this limited resource into packages, subject to audience collecting constraints.

Each TV break is characterized by its forecasted audience and by the number of advertisement messages it can contain (all messages having equal duration). Instead of selling these spots[1] one by one, the marketing department of the TV channel wants to prepare several packages of spots. For each package they define its price, size (number of spots) and "shape". This shape is a hierarchical time-zones decomposition of the

[1] What we call "spot" is a slot of 30 seconds during which an advertisement can be broadcasted.

week (week-end, prime-time, etc□) with minimum and maximum number of spots on each zone. Each package is also assigned an audience **requirement** i.e. a minimum total number of spectators (sum of forecasted audiences). Finally no package can have two spots in the same TV break.

The TV-BREAK PACKING PROBLEM (TVBP) consists in building all these packages, satisfying all size and shape constraints and maximizing the total revenue of those fulfilling their audience requirement.

With n the number of packages and m the number of TV breaks, we note q_i, r_i and g_i the size, audience requirement and price of package i and c_j and a_j the capacity and audience of TV break j. There are p_i time-zones for package i. Each zone $z_{i,k}$ (k^{th} zone of package i), is a subset of TV breaks and, for the sake of readability of the General Condition of Sales, two different zones of a package are always either non-overlapping or included in each other (**2**). Finally $q_{i,k}^{min}$ and $q_{i,k}^{max}$ are the minimum and maximum number of spots of package i in zone $z_{i,k}$ (**1**). The composition of packages is given by binary variables X_{ij}, equal to 1 when a message of package i will be broadcasted during TV break j (0 otherwise). Finally a binary variable Y_i is attached to each package: Y_i equals 1 if and only if package i contacts the required number of spectators, in which case its price g_i can be counted in the global revenue.

NAME: TV-BREAK PACKING PROBLEM

INSTANCE: Two integers n and m. Six vectors of non-negative integers q,p,r,g of length n and c,a of length m. n collections of triplets $\{(z_{i,k}, q_{i,k}^{min}, q_{i,k}^{max}) \mid k \in [1,p_i]\}$, such that

$$\forall i \le n,\ \forall k \le p_i \qquad z_{i,k} \subseteq [1,m] \quad and \quad 0 \le q_{i,k}^{\min} \le q_{i,k}^{\max} \tag{1}$$

$$\forall\ i \le n,\quad \forall\ k,l \le p_i \qquad z_{i,k} \cap z_{i,l} \in \{\varnothing, z_{i,k}, z_{i,l}\} \tag{2}$$

OBJECTIVE: **Find** X:$[1,n]\times[1,m]\to\{0,1\}$ and Y:$[1,n]\to\{0,1\}$ **such that**

$$\forall i \le n \qquad \sum_{j \le m} X_{ij} = q_i \tag{3}$$

$$\forall j \le m \qquad \sum_{i \le n} X_{ij} \le c_j \tag{4}$$

$$\forall i \le n,\ \forall k \le p_i \qquad q_{i,k}^{\min} \le \sum_{j \in z_{i,k}} X_{ij} \le q_{i,k}^{\max} \tag{5}$$

$$\forall i \le n \quad \sum_{j \le m} a_j X_{ij} \ge r_i Y_i \tag{6}$$

$$\textbf{Maximizing} \sum_{i \le n} g_i Y_i \tag{7}$$

Since the capacity of each TV-break cannot be exceeded, TV-break j cannot appear in more than c_j different packages (**4**). Each package must have the correct size (**3**) and satisfy shape constraints (**5**), that is to say that for each zone $z_{i,k}$, the number of TV-breaks selected by package i must belong to $[q_{i,k}^{min}, q_{i,k}^{max}]$. When a package fulfills its audience requirement its Y_i variable can be set to 1 in equation (**6**) and its price g_i can be added to the objective function (**7**).

It is important to note that all packages must be built, even if some audience constraints are violated. In other words we consider that package rejection is forbidden in our model: if a package has insufficient audience the final decision for this package (removal, discount, combination with another package□) belongs to the marketing department. In the rest of this paper we will focus on feasible instances of the problem, namely instances having integer solutions at least with $Y_i=0\ \forall\ i$.

2.2. Complexity

Proposition 1**.** TV-BREAK PACKING *is NP-hard in the strong sense.*

Proof. Let us consider the NP-complete 3-PARTITION problem [Garey & Johnson 1979]**.**

INSTANCE: A finite set X of *3n* positively weighted elements such that:

$$\forall i \in [1,3n],\ \frac{3\overline{w}}{4} < w(i) < \frac{3\overline{w}}{2} \qquad (8)$$

where *w(i)* is the weight of the i^{th} item and $\overline{w}$ is the average weight.

QUESTION: Can X be partitioned into *n* disjoint subsets of equal weight $3\overline{w}$? (note that equation (**8**) enforces these subsets to contain exactly three elements).

With any instance of 3-PARTITION, it is possible to associate an instance of *TVBP* by the following polynomial transformation.

- $m=3n$,
- $\forall\ i \le n\ q_i=3, r_i=3\overline{w}, p_i=0, g_i=1$,
- $\forall\ j \le 3n\ c_j=1, a_j=w(j)$,

The original 3-PARTITION instance is feasible if and only if the optimum of the associated *TVBP* instance is n. ■

2.3. Underlying flow structure

Thanks to property (**2**), equations (**3**),(**4**) and (**5**) (namely size, capacity and shape constraints) can be modelled by a flow [Ahuja et al. 1993] in the following graph, illustrated in **Fig. 1**.

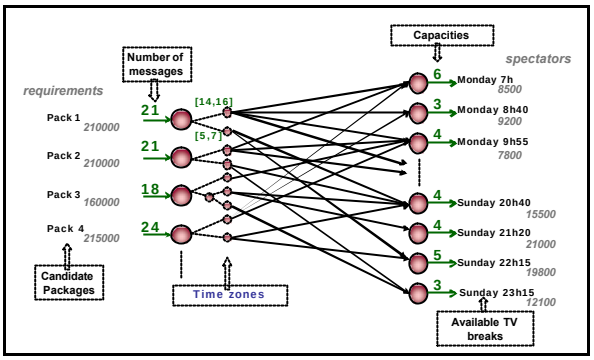

Fig. 1. Flow model

Each package i is a source node for q_i units of flow. One node is also associated to each time-zone $z_{i,k}$ with a single incoming arc of minimum and maximum capacities $q_{i,k}^{min}$ and $q_{i,k}^{max}$. The origin of this arc is the node representing the smallest zone containing $z_{i,k}$ if any, or the i^{th} source node otherwise. The uniqueness of this incoming arc is ensured by (**2**) thus zone nodes form a tree for each package. Leaves of these "shape trees" are linked with nodes representing contained TV breaks by arcs of capacity 1. Each of these TV break nodes has a single outgoing arc of capacity c_i toward a common sink. Note that this flow model allows determining the feasibility of the problem in polynomial time. This underlying flow structure also advocates for mathematical programming approaches (section 3) or for the use of a flow constraint in a CP model (section 4.3).

Concerning audience constraints (6) they are linear equations whose satisfaction brings a revenue g_i $\forall i \leq n$. As pointed out in previous section, these constraints are responsible for the NP-hardness of the problem.

3. Mathematical Programming

Linear Programming and Lagrangian Relaxation can be used to build solutions of the TVBP or to compute upper bounds of the maximal revenues. We first focus on the heuristic use of continuous or lagrangian relaxations to build feasible solutions. Then various upper bounds are considered, including those obtained by a lagrangian relaxation of capacity constraints.

3.1. Building solutions from relaxations

The linear model defined by equations (**3**) to (**7**) involves $O(nm)$ binary variables. A commercial MIP solver like Xpress-MP manages to optimally solve the smallest instances of our TVBP benchmark but fails to provide feasible solutions for many other

instances[2]. Therefore we use a greedy rounding procedure to build feasible packages from fractionary ones.

```
Integral(i) → VRAI si Y_i ∈ {0,1} et ∀j X_ij ∈ {0,1}

greedy →
    while ∃ i | (Y_i ≠ 0 and not(integral(i)))
        Y_i := 1
        let ε := 0 in
            while not(integral(i))
                let R := {j | Xij ∈ [1- ε,1[ } in
                    if R = ∅ increase ε
                    else for j in R X_ij := 1
                solveLP()
                if noLPSolution() backtrack up to statement « Y_i := 1 » and set Y_i := 0
    solveMIP()
```

Fig. 2. Greedy rounding algorithm

In this loop, the choice of the fractional package *i* to which the rounding procedure will apply is based on the fractional Y_i value, its reduced cost, the size of the package and its gain. Once all packages have been considered the residual problem *solveMIP()* merely consists in finding a feasible flow for discarded packages (Y_i=0).

An alternative approach consists in dualizing requirements constraints (equations (**6**)) introducing n non-negative lagrangian multipliers μ_i. The resulting lagrangian function to be maximized reads as follows:

$$\textbf{Maximize} \sum_{i \le n} g_i Y_i + \sum_{i \le n} \mu_i \left(\sum_{j \le m} a_j X_{ij} - r_i Y_i \right) \qquad (\mathbf{9})$$

For a given vector μ, each Y_i is only determined by the sign of $(g_i - \mu_i r_i)$. As for X_{ij} variables their computation is equivalent to solving a maximum cost flow where each arc $i \rightarrow j$ has cost $\mu_i a_j$. The intuitive interpretation of this relaxation is that arcs of packages failing to fulfill their audience requirement will be subsidized by the corresponding μ_i factor. This relaxation can be used to obtain primal solutions by « freezing » techniques [Rottembourg 1999]. Indeed the primalization of each dual solution is straightforward: it merely consists in setting some Y_i variables to 0 to ensure the satisfactions of constraints (**6**). Therefore we can fix packages one by one with the following algorithm in order to obtain a good primal solution.

```
freeze →
    A = {1, 2, … n}    // A is the set of « non-frozen » packages
    while A ≠ ∅
        solveLagrange(A) // find the Lagrangian optimal with packages outside of A constrained to keep their current solution
        let S = {i ∈ A | Σ a_j X_ij ≥ r_i} in
            if S = ∅ fin
            else A := A - {argmin_S(Σ aj Xij)}    // « freeze » the satisfied package with the smallest excess of audience
```

Fig. 3. Freezing algorithm

Lagrangian primalization (LAGp) and LP rounding (LPr) are compared in **Table 2** (page 16). We note that LAGp is three times slower than LPr and that the obtained

[2] See computational results in section 6.1

average qualities are similar (respectively 76% and 78% of the best known upper bound). However LAGp is more robust in the sense that the relative difference never exceeds 30% when LPr is better, whereas it can reach 400% when LAGp performs better.

3.2. Computing Upper Bounds for the TVBP

A first way to compute upper bounds for the TVBP consists in using the linear model defined in section 2.1. This linear bound can be tightened via the addition of knapsack inequalities like those that are automatically computed by a solver like Xpress-MP. It is also possible to perform a complete branch and bound on Y_i variables, relaxing only integrality constraints on X_{ij} variables. It shall be noted that the lagrangian relaxation of audience requirements presented in the previous paragraph is ineffective for the computation of stronger bounds than the trivial linear bounds. Indeed integrality constraints have no incidence on the lagrangian sub-problem. The consequence of this *Integrality Property* [Geoffrion 1974] is that the obtained upper bound will equal the continuous optimum of the linear model.

We consider here the dualization[3] of capacity constraints ([Frangioni 1997], [Gendron et al. 1999]). Relaxing equation (**4**) introducing one lagrangian multiplier λ_j for each TV-Break j yields the following lagrangian function:

$$\sum_{i \le n} g_i Y_i + \sum_{j \le m} \lambda_j \left(c_j - \sum_{i \le n} X_{ij} \right) \qquad (\mathbf{10})$$

For any λ, maximizing this function amounts to maximizing for each package $i \le n$ the following objective

$$\textbf{maximize } g_i Y_i - \sum_{j \le m} \lambda_j X_{ij} \textbf{ subject to } \text{equations } (\mathbf{3}), (\mathbf{5}) \text{ and } (\mathbf{6}) \qquad (\mathbf{11})$$

Observation 1. *Relaxation* (10) *is stronger than the continuous relaxation of the linear program (strictly on some instances).*

The lagrangian dual optimum is known to be as least as tight as the linear optimum [Geoffrion 1974]. The following example establishes that it is strictly stronger on some instances of the TVBP. For instance with $n=m=2$, $g_1=g_2=1$, $q_1=q_2=1$, $r_1=r_2=2$, $a_1=3$, $a_2=1$, $c_1=c_2=1$, the continuous solution $Y_1=Y_2=1$ and $\forall\, i,j\ X_{ij}=0.5$ satisfies all the constraints and has value 2, whilst the lagrangian upper bound obtained with $\lambda_1=1$ and $\lambda_2=0$ is 1. $(\max\{(Y_1 + Y_2) + (1 - (X_{11}+X_{21}))\} = 1$ because equation (**6**) requires $Y_i \le X_{i1}$ since the audience requirement cannot be fulfilled without the first TV-break (whose audience is 3). More precisely the lagrangian bound is the continuous optimum of function (**7**) subject to constraints (**4**) over the Cartesian product of convex hulls associated to each package. ■

[3] Identical bounds would be obtained using vectors $(Y_i, X_{i1}, \ldots, X_{im})$ satisfying (**3**), (**5**) and (**6**) as columns in a Dantzig-Wolfe decomposition.

Observation 2. *The langrangian sub-problems* (11) *are tractable.*

When Y_i=0, equation (**6**) has no impact and the problem is a minimum cost flow in a tree where arc towards leaves have cost λ_j. On the contrary the case $Y_i = 1$ is NP-hard since the minimization of $\sum \lambda_j X_{ij}$ subject to $\sum a_j X_{ij} \geq r_i$ (equation (**6**)) is a knapsack problem. Nonetheless, the standard pseudo polynomial dynamic programming approach can be adapted to this special knapsack, taking into account constraints (**3**) and (**5**).

Let us assume, without loss of generality, that TV-breaks are sorted such that each zone is an interval. We define a weight $\alpha(j) = \min(r_i, \sum\{\lambda_j X_{ij} \mid j \leq j\})$, $\eta_0(j) = \sum\{X_{ij} \mid j \leq j\}$ and for each zone $z_{i,k}$ a counter $\eta_k(j) = \sum\{X_{ij} \mid j \in z_{i,k} \wedge j \leq j\}$. Let $S(j) = (\alpha(j), \eta_0(j), \eta_1(j), \ldots \eta_{pi}(j))$ be the state of a dynamic program, after having chosen the value of $X_{ij} \forall j \leq j$. When j is the last screen of zone k then $\eta_k(j)$ must belong to $[q_{i,k}^{min}, q_{i,k}^{max}]$ and when $j=m$, both requirement and size constraints must be satisfied: $\alpha(m)=r_i$ and $\eta_0(m)=q_i$. From state $S(j) = (\alpha,\eta_0,\eta_1,\ldots)$ possible transitions are $(\alpha,\eta_0,\eta_1,\ldots) \rightarrow (\alpha,\eta_0,\eta_1,\ldots)$ of cost 0 and $(\alpha,\eta_0,\eta_1,\ldots) \rightarrow (\max(r_i,\alpha+a_{j+1}),\eta_0,\eta_1,\ldots)$ of cost λ_j. In fact the state of counters $\eta_0,\eta_1, \eta_2\ldots$ can be compressed using the following facts:

1. In state $S(j)$ only counters of zones including j are relevant,
2. when a zone with tight bounds $[q,q]$ is divided in two zones: $[q_1^{min},q_1^{max}]$ and $[q_2^{min},q_2^{max}]$, counters associated to zones z_1 and z_2 are useless: it is sufficient to restrict allowed values for the main counter η_0 at the end of z_1 to the interval $[\max(q_1^{min},q - q_2^{max}), \min(q_1^{max},q - q_2^{min})]$,
3. when a zone with bounds $[q^{min},q^{max}]$ is divided into p subzones with bounds $[q_1^{min}, q_1^{max}],[q_2^{min}, q_2^{max}] \ldots [q_p^{min}, q_p^{max}]$, the value of its counter η after the i^{th} subzone necessarily belong to the following interval: $[\max(\sum_{j\leq i} q_j^{min}, q^{min}-\sum_{j>i} q_j^{max}), \min(\sum_{j\leq i} q_j^{max}, q^{max} -\sum_{j>i} q_j^{min})]$.

With these properties the maximum number of states needed to represent counters is smaller than 70 on all the instances of our benchmark. Finally the total number of possible transitions in our dynamic program is smaller than 10^9, a huge but tractable size for modern computers. In practice we use a MIP program to solve this lagrangian sub-problem. ■

Observation 3. *Upper bounds can be improved combining regrets for each package with current lower and upper bounds.*

Maximizing this sub-problem for both possible values of Y_i (obtaining $opt_i(v)$ for $v \in \{0,1\}$) we can define the reduced cost $\rho_i(v) = \max(0, opt_i(1-v) - opt_i(v))$, representing the loss that would result if Y_i was to be set to its non-preferred value. The following "additive" bound can then be extracted and subtracted from the upper bound: $minloss = \min\{\sum_{i\leq n}\sum_{v\in\{0,1\}} \rho_i(v) Y_i$ subject to $lb \leq \sum_{i\leq n} g_i Y_i \leq ub\}$ where lb and ub denote respectively the current best known solution and upper bound[4] to the problem. These regrets can also be used to filter Y_i values in a branch and bound exploration: value v for Y_i can be removed if $\rho_i(v) > ub-lb$. ■

The following table reports upper bounds obtained by linear programming (LP) and by our lagrangian relaxation of capacity constraints (LAG) on a benchmark of 20 in-

[4] Note the virtuous cycle: when the upper bound is improved it can be used to make it even stronger.

stances, available from *http://e-lab.bouygues.com*. These instances involve from 4 to 100 packages with up to 1500 messages and 500 TV-breaks. As for shape constraints, the average number of zones per package (column $\bar{p}$) varies from 9 to 25 and the nesting depth of these zones is usually 2 or 3 and exceptionally 4. The pure lagrangian bound is better than the linear one on 10 instances (in bold font). The following pair of columns compares the LP model enriched with Xpress-MP automatically generated cuts with the lagrangian relaxation strengthened by the use of regrets, both bounds being rounded to the largest smaller subset sum of $\{g_i \mid i \le n\}$. Finally we improve both bounds by a branch and bound limited to selection variables (Y). The last column reports relative gaps with respect to best known solutions (cf. **Table 2**).

Table 1. Upper Bounds

					Pure Models		*Enriched Models*		*Branch& Bound on Y*		*Gap*
Bench	**∑qi**	**n**	**m**	$\bar{p}$	**LP**	**LAG**	**LP**	**LAG**	**LP**	**LAG**	
A1	64	4	208	14	31139	**29 735**	29 600	29 600	29 600	29 600	**0.0%**
A2	544	37	414	13	262966	262 966	262 966	262 966	262 966	262 966	0.8%
A3	1224	73	458	15	590934	590934	588 465	588 465	588 465	588 465	2.1%
A4	1492	84	456	15	730298	730298	730 298	730 298	730 298	730 298	0.3%
B1	85	8	167	9	13474	**12 300**	12 300	12 300	12 300	12 300	**0.0%**
B2	262	16	167	11	44100	**43 350**	42 600	42 600	42 600	42 600	**0.0%**
B3	325	21	172	11	52800	52 800	52 800	52 800	52 800	**52 100**	**0.0%**
B4	448	21	170	13	79000	**78 972**	79 000	**78 300**	79 000	**78 300**	1.8%
C1	68	4	157	21	18443	**16 756**	18 300	**15 300**	15 300	15300	**0.0%**
C2	108	7	180	20	31404	**29 617**	28 200	28 200	28 200	28200	**0.0%**
C3	457	31	218	20	127900	127 900	127 900	127 900	127 900	**127200**	**0.0%**
C4	809	50	219	20	238253	**237 186**	237 600	**235 300**	235 300	235300	3.2%
D1	138	13	286	19	36821	**32 300**	32 600	**32 300**	32 600	**32300**	**0.0%**
D2	691	52	286	23	185800	185 800	185 800	185 800	185 800	**184700**	3.9%
D3	810	56	286	25	216400	216 400	216 400	216 400	216 400	216400	3.8%
D4	916	63	286	25	244850	244 854	244 850	244 850	244 850	244850	3.6%
E1	159	16	224	17	28039	**25 130**	25 130	25 130	25 130	25130	**0.0%**
E2	902	96	291	16	163540	163 540	163 540	163 540	163 540	163540	3.9%
E3	983	99	291	16	179323	179323	179 220	**179 040**	178 080	**176580**	7.1%
E4	1082	100	291	18	196600	**196 599**	195 940	195 940	195 940	**194140**	4.1%
	Average CPU time				30s	15mn	2mn	15mn	5mn	3 hours	

The conclusion of this study is that the polyhedron of our lagrangian relaxation is tighter (strictly on 5 instances) than the one of the enriched linear model. Both models produce better bounds when included in a limited tree search. In order to improve these bounds, valid inequalities involving several packages should be design (since mono-package facets are implicit in our lagrangian formulation). Concerning CPU times, our experiments showed that bundle methods (available in Artelys Dualis library [Triadou et al. 2003]) were up to 10 times faster[5] than a naïve sub-gradient algorithm on these instances. Finally a good initial value for lagrangian multipliers is $\lambda_j = \varepsilon / a_j$ with $\varepsilon > 0$ (tend to avoid precious TV-breaks unless necessary).

[5] Another advantage of these methods is that they end with a convex optimality proof within a certain window.

4. Constraint Programming and Local Search

4.1. Constraints, support tuples and local search

Constraint Programming [Montanari 1974] is a problem-solving paradigm in which a problem is represented as a set of variables and constraints (see formal definitions in section 4.2). A constraint maintains the domain of all attached variables with a filtering algorithm removing values that are inconsistent with the mathematical meaning of the constraint. Each such inference on a variable can trigger propagation of other constraints of this variable. This constraint propagation mechanism allows detecting infeasibilities or reaching a fix point with restricted domains. From a branch and bound point of view, Constraint Programming is indeed a pruning technique.

It shall be noted that constraint propagation is based on *infeasible* values. The filtering algorithm of a constraint aims at removing infeasible values from the domain of its variables, i.e. values belonging to no tuple of the relation consistent with current domains. When all such inconsistent values are detected, the filtering algorithm is said to be complete. In the past few years, in order to perform more accurate filtering on rich n-ary constraints, several global filtering algorithms have been developed that are usually based on OR polynomial algorithms. Most of these *operationally global* constraints [Bessière & Van Hentenryck, 2003] maintain a *feasible* tuple consistent with current domains of variables. When no such support tuple exists, the constraint fails (detects a contradiction), otherwise it is used by the filtering algorithm. For instance the reference matching of the *AllDifferent* constraints [Régin 1994] ensure the feasibility of the constraint, and the strongly connected component decomposition based on this matching offers complete filtering.

The support tuple maintained by a constraint can be any element of the constraint support set, whereas some tuple may be much closer to "full" feasibility (of the whole problem) than others. Besides modifying the support tuple of a constraint is generally much easier than finding one. For instance moving from one matching to another (*AllDifferent* constraints) is just exchanging two arcs. Likewise, modifying a feasible flow requires no advanced knowledge of flow algorithms literature (c.f. 4.4). To the best of our knowledge CHOCO's Flow constraint was the first to give access to its internal reference flow, used as an oracle in [Benoist et al. 2002]. While keeping internal sophisticated OR algorithms in the constraint black box, we claim that constraint programmers can offer safe[6] APIs for reading or modifying the support of a constraint, allowing "moves" from a valid support to another.

Therefore after the presentation of a set of definitions and the description of a CP model for the TVBP centered on a flow constraint, we propose in paragraph 4.4 a local search approach to the TVBP based on local modifications of a feasible flow. This support tuple of a main constraint is the central element of the *Branch&Move* hybridization described in section 5.

[6] Making sure that no invalid modification is accidentally performed.

4.2. Definitions

Given K an ordered set and $D = D_1 \times D_2 \times \dots D_n$ with $D_i \subseteq K$ for all $i \in [1,n]$ (domains of variables), we define the following objects:

1. **Constraint**[7]**:** a *constraint* R is a relation on K^n ($R \subseteq K^n$)**.**
2. **Support set:** the *support set* of relation R on D is $supp(R,D) = R \cap D$. Omitting D, we will note the support set of a constraint R as $supp(R)=supp(R,D)$ with D equal to current domains.
3. **Support tuple:** a support of R is a tuple $x \in supp(R)$
4. **Constraint Satisfaction Problem:** A constraint satisfaction problem on K is a triplet (n,D,P) where P is a collection of constraints $P=\{R_1,R_2 \dots R_m\}$ on K^n.
5. **Solutions:** Solutions of P are $sol(P) = R_1 \cap R_2 \cap .. R_m \cap D$.
6. **Potential:** With δ_K a distance on K, we define the potential $\Delta(R,x)$ of constraint R with respect to tuple $x \in K^n$ as the L_1 distance from x to the current support set of R.

$$\Delta(R,x) = \min_{y \in supp(R)} \sum_{i \le n} \delta_K(x_i, y_i) \qquad (12)$$

Note that $\Delta(R,x)$ equals 0 if $x \in supp(R)$ and $+\infty$ if $supp(R)=\varnothing$. This potential literally measures the distance to feasibility of a tuple x for constraint R. For a binary constraint R involving two variables of domains D_1 and D_2, $\Delta(R,x)$ can be computed by enumeration in complexity $O(|D_1| \times |D_2|)$. Moreover, the potential of a linear constraint like $\sum X_i \ge X_0$, merely equals $max(0, x_0 - \sum x_i)$ (slack variable). When a exact computation of $\Delta(R,x)$ would be too costly, the distance to a feasible tuple of R (empirically close) is an upper bound of $\Delta(R,x)$.

7. **Corrective decision:** A corrective decision for a conflicting pair R,x (a pair with $\Delta(R,x)>0$) is a constraint A such that $x \notin A$ and $A \cap supp(R,D) \neq \varnothing$. In other words it is a constraint discarding x whilst consistent with R. A non-empty support for R is sufficient to ensure its existence.
8. **Neighbourhood:** A neighbourhood structure for R is a function $N_{R,D}$ associating to each support tuple x of R a subset $N_{R,D}(x) \subset supp(R,D)$ such that $x \in N_{R,D}(x)$.
9. **Selection:** A function *move* defined on $supp(R,D)$ is a selection function for neighbourhood $N_{R,D}$ if and only if $move(x) \in N_{R,D}(x)$.
10. **Improvement:** Given a strict order $<$ on K^n, *move* is an *improving* selection function if and only if $\forall\, x,\ move(x) = x \vee move(x) < x$. For instance the *potential order* $<_P$ defined by $x <_P y \Leftrightarrow \sum_{R \in P} \Delta(R,x) < \sum_{R \in P} \Delta(R,y)$ is a strict order on K^n. For optimization problems a second criteria based on the objective function can be added.
11. **Descent:** A descent from $x \in supp(R,D)$ is the iteration of an improving selection function *move* until a fix point is reached[8]:

    descent(x) → **while** (*move*(x) ≠ x) x := *move*(x), **return** x

Example with relations on $K^n = \{0,1,2\}^2$:

- $D_1 = D_2 = \{0,1\}$, $R = \{(0,0),(0,1),(1,0),(2,2)\}$,.

[7] Without loss of generality we extend constraints of smaller arity to relations on K^n.

[8] such a fix point is necessarily reached since K^n is finite and $<_P$ is a strict order.

- The *support* set of R is $supp(R,D)=R\cap(D_1\times D_2)=\{(0,0),(0,1),(1,0)\}$
- $(0,1)$ is a *support* of R (among the three possible ones).
- With $R'=\{(0,0),(1,1),(2,2)\}$ a second relation on $\{0,1,2\}^2$, solutions of problem $P=\{R,R'\}$ are $sol(P)=R\cap R'\cap(D_1\times D_2)=\{(0,0)\}$.
- The potential of R with respect to tuple $(1,1)$ is $\Delta(R,(1,1))=1$ because the closest support is $(0,1)\in supp(R,D)$ and $\delta((0,1),(1,1))=1$.
- A possible corrective decision for pair $R,(1,1)$ is $A=\{(0,x)\ \forall\ x\}$ since $(1,1)\notin A$ and $A\cap supp(R,D)=\{(0,0),(0,1)\}\neq\emptyset$.
- A possible neighbourhood for R is $N_{R,D}$: $(x,y)\rightarrow\{(x,0),(0,y),(x,y)\}$.
- For this simple neighbourhood, *move:* $(x,y)\rightarrow(0,0)$ is a possible selection function since $(0,0)$ belongs to the neighbourhood of any tuple of $supp(R,D)$.
- This selection function is strictly improving with respect to the potential order $<_P$, since $(0,0)$ is the only solution of P.

4.3. Constraint Programming Model for the TVBP

Our CP model for the TVBP uses the same variables and domains are our MIP model (section 2.1). However we use the following global constraints rather than linear constraints in order to ensure stronger propagations.

The underlying flow problem described in 2.3 can be embedded in a single flow constraint. This constraint [Benoist et al. 2002] ensures local consistency around each node propagating flow conservation rules. It also maintain a feasible flow consistent with current domains (using a highest-label preflow-push algorithm), throwing a contradiction as soon as no such support exists. It shall be noted that this support encompasses all decisions variables of the problem: once flow variables are set, setting Y_i to their resulting upper bounds (if consistent) leads to a feasible (but not necessarily optimal) solution.

Linear constraints would be semantically sufficient to model requirements equations (**6**), but would poorly interact with (3). For instance if m=5, a=[30,11,6,4,1] and q_1=2, the initial upper bound of r_1Y_1 would be 30+11+6+4+1=52. Therefore we introduce a complete filtering algorithm instead, detecting the exact upper bound and necessary or impossible TV-breaks. For instance the best upper bound in our example is 30+11=41, and event $r_1Y_1\geq 20$ triggers X_{11}=1. This constraint (selection of the q best objects among m) is similar to the one described in [Fahle & Sellmann 2002] for 0-1 knapsack constraints, without the difficulty of considering the last critical item (since all objects have equal sizes). The amortized complexity of this cost-based filtering equals the number of involved 0-1 variables.

Finally our CP model consists of a single flow constraint covering equations (**3**), (**4**)(**5**) and a set of n trivial knapsack constraints covering equations (**3**) and (**6**). As for the optimization criteria, it is modeled with a variable whose initial domain is $[0,\sum g_i]$ and linked with the linear combination (**7**) by an equality constraint.

4.4. Local moves for the TVBP

Our improvement process is based on 1-opt and 2-opt moves. The support flow gives a certain number of spectators to each package. Some packages receive more spectators than their objective, others not. Thus we try to give good TV breaks (i.e. those with many spectators) to non satisfied packages possibly through exchanges with satisfied packages. For $i \leq n$, we note Γ_i the number of spectators granted to package i in the support flow. When $\Gamma_i < g_i$, the potential of the i^{th} audience constraint with respect to the support flow is g_i-Γ_i.

Given package i and TV-breaks α and β, we name *1-opt* the following modification of the support flow. Remove one unit of flow on the path from source i to node α and add one unit of flow on the path from souce i to node β. This move is feasible if the obtained flow still respects capacities. It is an improving move if $\Gamma_i < g_i$ (before the move) and $a_\beta > a_\alpha$, that is to say if it decreases the potential of package i.

Given packages i and j and TV-breaks α and β, we name *2-opt* the following modification of the support flow. Remove one unit of flow on paths from source i to node α and from source j to node β; add one unit of flow on the path from source i to node β and from source j to node α. This move is feasible if the obtained flow still respects capacities. It is an improving move if $\Gamma_i < g_i$ (before the move), $a_\beta > a_\alpha$ and $\Gamma_j - g_j > a_\beta - a_\alpha$, that is to say if it decreases the potential of package i whilst package j remains satisfied.

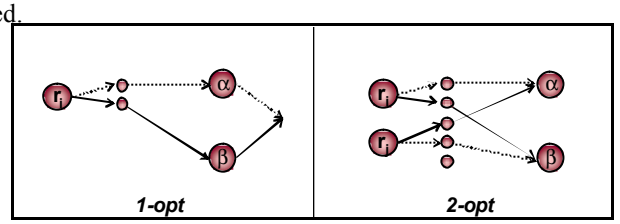

Fig. 4. Local moves

Both 1-opt and 2-opt moves can be specified to the flow constraint as a list of arcs forming a cycle in the non-directed graph. Scanning the cycle in the direction of the first arc, one unit of flow is added or removed to arcs depending on their direction, checking on the fly that capacities remain respected. This API is available in CHOCO's flow constraint, with an additional parameter for the desired amount of circulating flow. It defines a neighborhood structure for the flow constraint. Cycles for 1-opt and 2-opt moves are illustrated on **Fig. 4**, with one unit of flow removed on dotted arcs and added on others.

5. The Branch and Move Algorithm

For problems where a principal constraint can be identified (namely an underlying structure for which a polynomial algorithm can be embedded in a constraint), we propose a support-guided branching scheme and an associated cooperation between Constraint Programming and Local Search. We name this procedure *Branch and Move*. In this section we define this algorithm and apply it to the TVPB.

5.1. Definition and motivations

Let R_0 be the main constraint of a problem P and x an element of its support. If the potential of x with respect to all other constraints is null ($\forall R \in P, \Delta(R,x)=0$) then x is a solution of P. Otherwise we decide to select among all additional constraints the one with highest potential and to branch on an associated corrective decision[9]. More precisely we select R maximizing $\Delta(R,x)$ and A a corrective decision for R,x. Then we successively consider sub problems $P \cup A$ and $P \cup \bar{A}$, with $\bar{A}=K^n-A$ (complementary decision). In order to make this branching strategy more efficient, the global appropriateness of the support tuple x is improved by local search before each choice: according to a neighbourhood structure suitable for constraint R_0, a descent procedure is applied on x. The resulting *Branch and Move* algorithm reads as follows:

```
solve(P) →                                          // returns true if P is feasible, false otherwise
  if not (propagate(P)) return false,               // constraints propagation, returns false in case of inconsistency
  else
    let x := descent(getSupport(R0)),               // improvement of the support tuple of R0 by local search
        Rmax := argmax{Δ(R,x), R∈ P} in             // selection of the most unhappy constraint w.r.t. x
      (if (Δ(Rmax,x) = 0) return true               // if x satisfies all constraints it is a solution of P
       else
         let A = selectCorrective(Rmax,x) in        // selection of a corrective decision for Rmax
           return (solve(P∪A) or solve(P∪Ā))        // recursive exploration
```

Fig. 5. The Branch and Move algorithm

It is important to note that the improvement of the support by local search is completely non-destructive: it has no impact at all on current domains since only the support tuple is modified. Instead of jumping to another node of equal depth, modifying already instantiated variables as in *Incremental Local Optimization* [Caseau & Laburthe 1999], we modify the values taken by *remaining* variables in the *support tuple*. This absence of impact on current domains keeps the search tree complete whilst the embedded problem-specific local search algorithm avoids wasting time in a costly sub tree searching for a solution laying just a few moves far from current support tuple, or trying the ensure the satisfaction of constraints that are consistent with a nearby support tuple. The latter case points out that the improvement process helps identifying critical points: constraints remaining in conflict with the obtained support ("resisting to local moves") may be the most critical ones. The *Branch and Move* algorithm can be compared to the *Branch and Greed* [Sourd & Chrétienne 1999] and *Local Probing* techniques [Kamarainen & El Sakkout 2002] that are also based on heuristic computations at each node, searching for possible extension of the current partial assignement. From a more general point of view these algorithms belong to the family of CP/LS hybrids [Focacci et al 2003], [Focacci & Shaw 2002].

From Local Search point of view, the whole process can be seen as the systematic exploration of the support set of the main constraint. In this context, the initial im-

[9] This approach is directly inspired from what can be done in a MIP branch and bound when a continuous solution is found: a violated integrality constraint is selected, and a decision is taken to bring the LP solver to modify this best continuous solution accordingly.

provement process is a standard greedy descent starting from the support tuple. Now the local optimum escaping strategy significantly differs from tabu or simulated annealing paradigms. Instead of changing the solution by non-improving moves, the landscape itself is modified thanks to a corrective decision. The propagation of this decision removes many tuples from the landscape, including the current one. The filtering algorithm computes a new support tuple close to the discarded one. Then a new greedy descent is applied from this new starting point to reach the closest local optimum in this *new filtered landscape*. Since these decisions are embedded in a CP search tree, a complete enumeration of possible landscapes is performed yielding to an *exact* local search algorithm. When such a complete enumeration would be too costly, the CP framework gives us the opportunity to use well-tried partial tree search techniques developed in the last decade. Restricted Candidate Lists, discrepancy based search, credit search, barrier limit, etc… are likely to guide the enumeration through a diversified set of promising landscapes [Focacci et al 2003].

For a more comprehensive description of *Branch and Move* principles and related work, we refer the reader to [Benoist & Bourreau, 2003].

5.2. Applying the Branch and Move algorithm to the TVBP

Only improving moves are performed, starting with the pass of 1-opt moves. Packages with the largest negative $\Gamma_i - r_i$ (closest to 0) are served first. Symmetrically, packages with the largest positive $\Gamma_i - r_i$ are "robbed" first[10] during the 2-opt pass.

At each node of the search tree, this local search descent is performed and the resulting support flow is evaluated, summing the prices of satisfied packages ($\Gamma_i - r_i \geq 0$). If this evaluation is greater than the current best, this solution is registered. Then the least unsatisfied package is considered. Values 1 and 0 are successively tried for variable X_{ij}, where j is the best TV-break not given to package i in the support flow. Note that $X_{ij}=1$ is a corrective decision for the audience constraint of package i. However this constraint is not the one with highest potential as suggested in 5.1 since in this Weighted CSP context we prefer to fulfill one constraint rather than decreasing the potential of two constraints.

6. Conclusions

6.1. Computational results

In the following table we compare solutions obtained by solving approaches described in this paper. For the sake of readability, results are expressed as percentages of the best known upper bounds (obtained in section 3.2). Therefore 100 always de-

[10] Packages whose audience constraints cannot be satisfied ($Y_i = 0$) are treated as if their audience objective was null ($r_i = 0$). A third pass is also added to distribute spectators from the most unsatisfied packages to the least unsatisfied ones.

notes an optimal solution[11], and for open instances the relative gap between the best solution and the best upper bound is "100-bestSol". Bold font is used for the best solution on each line and for the names of closed benchs. Finally 300 seconds[12] have been granted to each algorithm (line "CPU" reports averages times for best solution).

Table 2. Comparative results

Bench	$\sum qi$	n	m	*MIP*	*CP*	*LS*	*LAGp*	*LPr*	*BM*
A1	64	4	208	**100**	**100**	66.9	59.5	52.0	**100**
A2	544	37	414	87.7	64.0	76.2	68.9	89.1	**99.2**
A3	1224	73	458	0	64.2	75.8	71.9	89.1	**98.0**
A4	1492	84	456	0	59.4	80.1	73.7	94.2	**99.7**
B1	85	8	167	**100**	**100**	85.4	**100**	20.3	**100**
B2	262	16	167	**100**	64.6	72.5	76.3	**100**	95.8
B3	325	21	172	**100**	52.8	65.3	71.6	84.6	98.5
B4	448	21	170	**97.2**	56.2	70.0	66.4	86.8	97.1
C1	68	4	157	**100**	60.1	4.6	**100**	79.7	**100**
C2	108	7	180	**100**	65.2	48.2	69.9	97.5	**100**
C3	457	31	218	**97.6**	65.7	66.8	73.1	94.7	**97.6**
C4	809	50	219	92.1	56.7	72.4	77.9	**96.0**	95.3
D1	138	13	286	**100**	**100**	75.4	**100**	24.0	**100**
D2	691	52	286	83.2	47.7	65.2	68.7	91.0	**96.3**
D3	810	56	286	0	51.5	75.3	66.5	90.9	**96.3**
D4	916	63	286	81.5	52.3	68.7	70.0	89.9	**96.6**
E1	159	16	224	**100**	77.8	**100**	**100**	19.1	**100**
E2	902	96	291	71	60.0	64.3	68.9	80.6	96.2
E3	983	99	291	0	62.9	73.7	70.0	89.0	**93.4**
E4	1082	100	291	0	64.2	70.1	70.8	90.1	**94.9**
		Average result		71	66	69	76	78	**98**
		Average CPU time		90s	50s	0.1s	100s	30s	13s

A commercial solver like Xpress-MP (column MIP) is very efficient to solve small instances but obtains poor results on larger instances, even sometimes no solution at all (on fives instances). LP-based greedy algorithms LAGp and LPr of section 3.1 perform better on those large problems. The CP column lists results obtained using the CP model of section 4.3 in a tree search, tending to give the best possible TV-break to the least unsatisfied package at each node. This approach is robust but never manages to prove optimality even when the best solution is found. Solutions "LS" result from the application of our greedy descent with the initial support flow as starting point. These local optimums have medium quality. No optimal solution is found even on small instances like C1.

Finally combining Constraint Programming and Local Search in our *Branch & Move* hybrid leads to a robust and efficient algorithm: the gap with the best known upper bound never exceeds 5%. Thanks to local search performed on the flow support and to our branching strategy, these excellent solutions are found very quickly (13s in average). Besides, the focus on critical packages (remaining non-satisfied after the LS pass) seems to be a very good first-fail heuristic. The resulting decrease of the number of nodes allows completing the search on four of the smallest instances whilst these

[11] This figure is underlined when the solving algorithm proves the optimality

[12] Note that granting 6 hours to Xpress-MP leads to better solutions for problems B4, C3, C4 and E4 (respectively 98.6, 100, 96.9, 96.1) while some others remain without any solution. This extended computation time in not acceptable in the operational context anyway.

optimality proofs were not obtained by an exploration based on the same CP model but deprived of the LS oracle.

This successful use of the support of the main constraint of a problem raises the issue of the generalization of this idea to problems where no such central constraint can be identified. In such cases, the support tuples of several constraints would have to be synchronized, possibly through lagrangian decomposition [Guignard & Kim, 1987].

6.2. Hard and easy extensions of the TVBP

A first easy extension of the TVBP consists in taking into account mutual exclusion between advertisements: for instance several car advertisements cannot coexist in a single TV-break. These protected commercial sectors (cars, perfumes, mobile phones, etc…) can be represented by colors in our model. In the list of candidate packages, the marketing department can mark some packages with one of these colors to specify the economic sector to which this package is intended to be sold. With k_i the color of package i (0 coding for "non-protected"), these additional constraints read as follows:

$$\forall j \le m, \forall \gamma > 0 \qquad \sum_{i \le n | k_i = \gamma} X_{i \to j} \le 1 \tag{13}$$

In our flow model (section 2.3) this is equivalent to the insertion of one collecting node per color before each TV-break node. Therefore all approaches listed in this paper apply to this case.

A second aspect ignored in this paper is audience segmentation. In practice, the forecasted audience of a TV-break is detailed by target groups and the audience requirement of different packages may be focused on different target groups. When each package is limited to one target group this extension tends to make the problem easier since a TV-breaks exchange (2-opt) between packages with different targets can increase the collected audience of *both* packages. However algorithms need to be adapted when several requirements constraints are allowed for each package (for instance guaranteed audiences on children *and* housewives).

Finally our model could also help the marketing department in the choices of prices for each package. Indeed the price of a family of packages depends on the audience constraint stated for these packages: if the guaranteed audience is set too high it will be difficult to satisfy (very few precious packages will be built), but if it is set too low the corresponding price will be too small to yield high revenues (many cheap packages). Finding the optimal price for each family of packages is a bi-level optimization problem whose second stage is the TVBP.

References

R.K. Ahuja, T.L. Magnanti and J.B. Orlin, 1993. *Network Flows : theory, algorithms and applications*. Prentice Hall.

N. Beldiceanu and E. Contejean, 1994. *Introducing global constraints in CHIP*. Mathematical and Computer Modelling, 20(12):p 97-123.

T. Benoist, E. Bourreau, 2003. *Improving Global Constraints Support by Local Search*. In COSOLV'03.

T. Benoist, E. Gaudin, B. Rottembourg, 2002. *Constraint Programming Contribution to Benders Decomposition: A Case Study*. In CP-02, pages 603-617.

C. Bessière and P. Van Hentenryck, 2003. *"To be or not to be...a global constraint"*. In Proceedings CP'03, Kinsale, Ireland, pages 789-794.

A. Bockmayr, N. Pisaruk, A. Aggoun, 2001. *Network Flow Problems in Constraint Programming*. In CP-01, pages 196-210.

E. Bourreau, 1999. Traitement *de contraintes sur les graphes en programmation par contraintes*. PhD thesis of University Paris 13.

Y. Caseau, F. Laburthe, 1999. *Heuristics for Large Constrained Vehicle Routing Problems*, Journal of Heuristics 5 (3), Kluwer.

V. Chvatal, 1983. *Linear programming*, Freeman.

T. Fahle, M. Sellmann, 2002. *Cost Based Filtering for the Constrained Knapsack Problem*. In Annals of Operations Research 115:73-94.

M.L. Fisher, 1981. The Lagrangian relaxation method for solving integer programming problems. Management Science, 27:1-18.

F. Focacci, F. Laburthe, A. Lodi, 2003. *Local search and constraint programming*. In Handbook of Metaheuristics pages 369-403, Kluwer.

F. Focacci and P. Shaw, 2002. *Pruning sub-optimal seach branches using local search*. In Proceedings of CP-AI-OR-02 pages 181-189.

A. Frangioni, 1997. *Dual ascent methods and multicommodity fow problems*. PhD thesis, Dipartimento di Informatica, Universita di Pisa.

M.R. Garey and D.S. Johnson, 1979. *Computers and intractability, a guide to the theory of NP-completeness*. New York: W. H. Freeman.

B. Gendron, T. Crainic, and A. Frangioni, 1999. *Multicommodity capacitated network de-sign*, in Telecommunications network planning (P. Soriano, and B. Sanso, eds.), Boston: Kluwer Academic Publisher (pages 1-19).

A.M. Geoffrion, 1971. *Duality in Nonlinear Programming*, SIAM Review, 13:1, 1-37.

A.M. Geoffrion, 1974. *Lagrangean relaxation for integer programming*. Math. Programming Stud. 2, 82-114.

M. Guignard and S. Kim, 1987. *Lagrangian decomposition: a model yielding stronger Lagrangian bounds*, Mathematical Programming, 39, 215-228.

O. Kamarainen, H. El Sakkout, 2002. *Local Probing Applied to Scheduling*. In CP02: 155-171.

U. Montanari, 1974. *Networks of constraints: fundamental properties and applications to picture processing*. Information Science, 7: 95-132.

J-C. Régin, 1994. *A filtering algorithm for constraints of difference in CSPs*. In AAAI 94, Twelth National Conference on Artificial Intelligence, pages 362-367, Seattle, Washington.

B. Rottembourg, 1999. *Une heuristique basée sur le dual Lagrangien pour un problème combinatoire de gestion de parc de matériel*. In Journées du groupe MODE, Orléans.

F. Sourd, P. Chrétienne, 1999, *Fiber-to-Object Assignment Heuristics.* In European Journal of Operational Research, 117.

M. Trick, 2001. *A dynamic programming approach for consistency and propagation for knapsack constraints*. Proceedings of CPAIOR-01 pages 113-124.

C. Triadou, C. Lemaréchal, J. Maeght and A. Renaud, 2003. *A bundle method for convex optimization: implementation and illustrations*. In ISMP 03, Copenhagen, Denmark.