General Bounding Mechanism for Constraint Programs

Minh Hoàng Hà[1], Claude-Guy Quimper[2], Louis-Martin Rousseau[1]

[1] Department of Mathematics and Industrial Engineering and CIRRELT, École Polytechnique de Montréal, C.P. 6079, Succursale Centre-ville, Montreal, QC, Canada H3C 3A7
{minhhoang.ha, louis-martin.rousseau}@cirrelt.net
[2] Département d'informatique et de génie logiciel and CIRRELT, Université Laval, Quebec, Canada
claude-guy.quimper@ift.ulaval.ca

Abstract. Integer programming (IP) is one of the most successful approaches for combinatorial optimization problems. Many IP solvers make use of the linear relaxation, which removes the integrality requirement on the variables. The relaxed problem can then be solved using linear programming (LP), a very efficient optimization paradigm. Constraint programming (CP) can solve a much wider variety of problems, since it does not require the problem to be expressed in terms of linear equations. The cost of this versatility is that in CP there is no easy way to automatically derive a good bound on the objective. This paper presents an approach based on ideas from Lagrangian decomposition (LD) that establishes a general bounding scheme for any CP. We provide an implementation for optimization problems that can be formulated with knapsack and regular constraints, and we give comparisons with pure CP approaches. Our results clearly demonstrate the benefits of our approach on these problems.

Keywords: Constraint Programming, Automatic Bounding, Lagrangian Decomposition, Knapsack Constraint, Regular Constraint.

1 Introduction

Constraint Programming (CP) is an efficient tool for complex decision problems arising in industry, such as vehicle routing, scheduling, and resource allocation. CP solvers essentially solve satisfiability problems, that is, they determine whether or not there exists a solution to a given model. When applied to optimization problems, most CP solvers solve a sequence of satisfiability problems, requiring at each step that the solution found improves on the solution found at the previous step. The search stops when no feasible solution can be found, proving that the previous solution was indeed optimal. The ability to compute bounds on the objective function (the optimization criterion) is crucial. It allows faster termination of the final subproblem (by proving infeasibility more

quickly), and it speeds up the solution of all the subproblems (because filtering and pruning become possible when a good bound on the objective is known). In all cases where CP has successfully solved optimization problems, the modeler had to implement good bounding mechanisms. The lack of a good general-purpose bounding scheme is one of the main drawbacks of CP in comparison with integer programming (IP).

In CP, the constraints are independent, and information is communicated solely through the *domain store*, the set of possible remaining values for each variable. During the search, each constraint looks at the domain store, determines whether a solution still exists, and filters out pairs of variables and values that no longer lead to feasible solutions. This decoupling of the problem in terms of the independent constraints allows us to use a simple and efficient combinatorial algorithm for each constraint. A similar concept is used in linear programming (LP): Lagrangian decomposition (LD) relaxes the link between difficult sets of constraints and introduces in the objective a penalty vector that acts as the glue linking the remaining relatively simple subproblems. In CP, we observe that there is no such glue for the constraints (or subproblems), making any default bound computation very weak. If we look at CP through LD glasses, we see that the relaxed constraints are actually a set of implicit constraints stating that any given variable should have the same value in all the different constraints in which it appears. Therefore, to apply LD techniques to a CP model, we must penalize that different constraints assume the same variable can take different values during search. Our goal is to develop a general bounding mechanism for CP that is compatible with every CP model and transparent to the user.

This paper is organized as follows. The next section reviews related work on decomposition techniques and CP, and Section 3 presents our general bounding approach. Section 4 evaluates the method on two optimization problems: a) the knapsack problem and b) the simple shift scheduling problem that can be modeled as a combination of many regular constraints. Finally, Section 5 summarizes our conclusions.

2 Related Work

We present here some background on LD and the global CP constraints investigated in this paper.

2.1 Lagrangian Decomposition

Decomposition techniques are an efficient approach for large-scale optimization problems. One of the most popular techniques is Lagrangian relaxation (LR), a natural approach that exploits the problem structure. It relaxes the "complicated" constraints and penalizes the constraint violation in the objective function. This simplifies the solution procedure since the resulting problem is easier to solve.

Several authors have used LR coupled with CP to solve optimization problems. Sellmann and Fahle [20] propose an algorithm for the automatic recording problem. They introduce two linear substructures, and they dualize the first and propagate the second. At convergence, they use the optimal dual information to propagate the first substructure. Ouaja and Richards [15] embed LR and CP into a branch-and-bound algorithm for the traffic placement problem. At each node of the search tree, CP is used to find a solution if one exists and to prove infeasibility otherwise, and LD indicates how close the solution is to optimality. Benoist et al. [2] propose hybrid algorithms combining LR and CP for the traveling tournament problem, which includes round-robin assignment and travel optimization. They show that LR provides not only good bounds to limit the search but also information to guide new solutions or efficient branching. Sellmann [19] investigates the theoretical foundations of CP-based LR and proves that suboptimal Lagrangian multipliers can have stronger filtering abilities than optimal ones.

A drawback of LR is that the problem loses its original structure since the complicated constraints are removed from the constraint set and penalized in the objective function. Moreover, the penalized constraints must generally be linear. This seriously restricts the application of LR in the context of CP where most global constraints have a nonlinear structure. For example, it is not easy to dualize some popular global constraints such as the all-different, global cardinality, and regular constraints. To overcome this restriction, Fontaine et al. [8] generalize LR to arbitrary high-level models using the notion of satisfiability (violation) degree. In this method, a satisfiability degree (or violation degree) is defined for each hard constraint and penalized in the objective function instead of in the constraint itself. The results show that LR coupled with CP can efficiently solve some classes of graph coloring problems.

Another way to avoid the penalization of nonlinear hard constraints is to use LD. This was first introduced by Guignard and Kim [10]; it is also called variable splitting. It creates "copies" of a set of variables responsible for connecting important substructures in the model, using one copy per substructure and dualizing the equality of a variable and its copy. Since the substructures of the original problem are retained, LD provides bounds that are always at least as tight as those from LR. The method is particularly useful if there are no apparent complicating constraints or the complicating constraints have nonlinear forms.

The research on integrating LD into CP is limited. To the best of our knowledge, only Cronholm and Ajili [5] have studied this. They propose a hybrid algorithm for the multicast network design problem. Their approach enables the propagation of all the substructures at every dual iteration and also provides strong cost-based filtering. We continue their theme with our attempt to use LD to construct an automatic bounding mechanism, thus making CP tools more efficient for optimization problems. To do this, we consider each global constraint in the CP model as a substructure, and we connect them through LD to pro-

vide valid bounds. These are used to prune the nodes in the search tree. We investigate two global constraints: knapsack and regular constraints.

2.2 Knapsack and Regular Constraints

In this subsection, we discuss knapsack and regular constraints and formulate the related optimization problems.

Knapsack Constraint Knapsack constraints are probably the most commonly used constraints for problems in mixed integer programming and CP. These linear constraints have the form $wx \leq W$ where W is scalar, $x = [x_1, x_2, ..., x_n]$ is a vector of n binary variables, and $w = [w_1, w_2, ..., w_n]$ is a vector of n coefficients.

Most CP solvers filter knapsack constraints in a straightforward manner: the domain reduction is based on interval arithmetic and simple bounding arguments. This filtering is fast and effective but requires an important branching in practice. Another knapsack filtering has been proposed by Trick [22], who uses a dynamic programming structure to represent the constraint and achieves hyper-arc consistency, thus determining infeasibility before all the variables are set. The filtering is based on a layered directed graph where the rows correspond to W values (0 through $|W|$), the columns represent variable indexes (0 through n), and the arcs represent variable-value pairs. This graph has the property that any path from (0, 0) to the nodes of the last layer corresponds to a feasible solution to the knapsack constraint.

The filtering can reduce the branching when dealing with the satisfiability version of the knapsack problem. This version, the market split problem, has no objective function (see Trick [22] for more information). An effective implementation is essential to reduce the computational time. In the next sections, we show that integrating the filtering proposed by Trick [22] with LD reduces both the branching and the computational time for optimization problems.

Fahle and Sellmann [7] introduce an optimization version of the knapsack constraint. Given a lower bound $B \in \mathbb{N}$ and an vector of n coefficients $p = [p_1, p_2, ..., p_n]$, the constraint enforces not only $wx \leq W$ but also $px \geq B$. Since achieving generalized arc consistency for this constraint is NP-hard, Fahle and Sellmann [7] introduce the notion of *relaxed consistency* for optimization constraints, and they use bounds based on LP relaxations for polynomial-time domain filtering. Using bounds with guaranteed accuracy, Sellmann [18] exploits an existing approximation algorithm to provide an *approximate consistency*. Katriel et al. [11] develop an efficient incremental version of the previous algorithm. Malitsky et al. [13] generalize the method of [11] to provide a filtering algorithm for a more general constraint, the bounded knapsack constraint, in which the variables are integer.

The knapsack constraint is the main component of the class of knapsack problems, which has many variants (see Kellerer et al. [12] for more details). The simplest is the 0/1 knapsack problem. Given a set of n items, where each item i has a weight w_i and a value p_i, we must determine which items to include

in the knapsack so that the total weight is less than or equal to a given limit W and the total value is maximized. In this paper, we focus on the multidimensional knapsack problem (MKP), which is as follows:

$$\text{Maximize} \quad \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_{ij} x_i \leq C_j \qquad j = 1, \ldots, m \tag{2}$$

$$x_i \in \{0, 1\} \tag{3}$$

Regular Constraint The regular global constraint is generally specified by using a (nondeterministic) finite automaton π, which is defined as a 5-tuple $\pi = (Q, A, \tau, q_0, F)$ where:

- Q is the finite set of states;
- A is the alphabet;
- $\tau : Q \times A \times Q$ is the transition table;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is a set of final states.

The constraint REGULAR$([X_1, \ldots, X_n], \pi)$ holds if the sequence of values of the variables $X_1, \ldots, X_n$ is a member of the regular language recognized by a deterministic finite automaton π. The recognition is confirmed if there exists a sequence of states $q_{i_0}, \ldots, q_{i_n}$ such that $q_{i_0} = q_0$, $(q_{i_k}, X_i, q_{i_{k+1}})$ is a transition in τ, and $q_{i_n} \in F$ is a final state.

Pesant [16] introduced a filtering algorithm for the regular constraint. It constructs a layered graph similar to that proposed for the knapsack constraint above, except that its rows correspond to states of the automaton. This approach was later extended to the optimization constraints SOFT-REGULAR [23], COST-REGULAR [6], and MULTICOST-REGULAR [14] to enforce bounds on the global cost of the assignment. The underlying solution methods compute the shortest and longest paths in an acyclic graph.

The regular global constraint is useful in modeling complex work regulations in the shift scheduling problem (SSP), since the deterministic finite automaton can specify the rules that regulate transitions in the sequence of activities [6, 17, 4, 3]. The problem consists in scheduling a sequence of activities (work, break, lunch, and rest activities) for a set of employees. We will investigate a version of the SSP in which only one employee is considered and the work regulations are so complex that modeling them requires many regular constraints. Let W be a set of work activities, T a set of periods, Π a set of deterministic finite automata expressing work regulations, and p_{ij} the profit for the assignment of activity $i \in W$ at period $j \in T$. We must assign one and only one activity for each period such that the transition of the activities over the periods satisfies the work regulations defined by the automata in Π and the total profit is maximized. To state the model for the SSP in CP, we use the element constraint

ELEMENT(INDEX, TABLE, VALUE), which holds if VALUE is equal to the INDEX of the TABLE, i.e., VALUE = TABLE[INDEX], to represent the assignment. Let X_j be the variable representing the activity assigned to period j. The problem is then:

$$\text{Maximize} \quad \sum_{j \in T} C_j \tag{4}$$

$$\text{subject to} \quad \text{REGULAR}(X, \pi_i) \quad \forall \pi_i \in \Pi \tag{5}$$

$$\text{ELEMENT}(X_j, p_{ij}, C_j) \quad \forall i \in W, j \in T \tag{6}$$

$$X_j \in W \quad \forall j \in T \tag{7}$$

$$C_j \in \mathbb{R}, \min_{i \in W} p_{ij} \leq C_j \leq \max_{i \in W} p_{ij} \quad \forall j \in T \tag{8}$$

where constraints (6) ensure that C_j is equal to p_{ij} if and only if $X_j = i$. This specifies the profit of each assignment.

3 Proposed Approach

3.1 Lagrangian Decomposition

We first recall the LD approach. Consider the problem of computing $\max\{z = c^\top x | C_1(x) \wedge C_2(x)\}$, where x is a set of variables, c is a set of coefficients, and $C_1(x)$ and $C_2(x)$ are two arbitrary constraints. One can obtain a relaxation, i.e., an upper bound on the solution, as follows:

$$\begin{aligned}
\max\left\{z = c^\top x \,\middle|\, C_1(x) \wedge C_2(x)\right\} &= \max\left\{c^\top x \,\middle|\, C_1(x) \wedge C_2(y) \wedge x = y\right\} \\
&= \max\left\{c^\top x + u^\top (x - y) \,\middle|\, C_1(x) \wedge C_2(y) \wedge x = y\right\} \\
&\leq \max\left\{c^\top x + u^\top (x - y) \,\middle|\, C_1(x) \wedge C_2(y)\right\} \\
&= \max\left\{(c^\top + u^\top)x \,\middle|\, C_1(x)\right\} + \max\left\{-u^\top y \,\middle|\, C_2(y)\right\}
\end{aligned}$$

In LD, a set of variables x is duplicated by introducing an identical set of variables y, and the difference $(x - y)$ is added to the objective function with a violation penalty cost $u \geq 0$, where u is the vector of Lagrangian multipliers. The original problem is then relaxed to two subproblems: one with C_1 and the other with C_2. The two subproblems can be solved separately. Solving the resulting programs with a given parameter u provides a valid upper bound on the original objective function. The non-negative multipliers u that give the best bound can be found by solving the Lagrangian dual. The decomposition can easily be generalized to m constraints:

$$\begin{aligned}
\max\left\{z = c^\top x \,\middle|\, \bigwedge_{i=1}^{m} C_i(x)\right\} &= \max\left\{c^\top x^1 \,\middle|\, \bigwedge_{i=1}^{m} C_i(x^i) \bigwedge_{i=2}^{m} x^i = x^1\right\} \\
&= \max\left\{c^\top x^1 + \sum_{i=2}^{m} u(i)^\top (x^1 - x^i) \,\middle|\, \bigwedge_{i=1}^{m} C_i(x^i) \bigwedge_{i=2}^{m} x^i = x^1\right\} \\
&\leq \max\left\{c^\top x^1 + \sum_{i=2}^{m} u(i)^\top (x^1 - x^i) \,\middle|\, \bigwedge_{i=1}^{m} C_i(x^i)\right\} \\
&= \max\left\{\left(c^\top + \sum_{i=2}^{m} u(i)^\top\right) x^1 \,\middle|\, C_1(x^1)\right\} + \sum_{i=2}^{m} \max\left\{-u(i)^\top x^i \,\middle|\, C_i(x^i)\right\}
\end{aligned}$$

This decomposition works well for numeric variables, i.e., variables whose domains contain scalars. In CP, we often encounter domains that contain non-numeric values. For instance, the regular constraint described in Section 2.2 applies to variables whose domains are sets of characters. In most applications, it makes no sense to multiply these characters with a Lagrangian multiplier. In this situation, we do not apply the LD method to the original variables X_i but instead to a set of binary variables $x_{i,v} \in \{0, 1\}$ where $x_{i,v} = 1 \iff X_i = v$. Rather than having a Lagrangian multiplier for each variable X_i, we instead have a multiplier for each binary variable $x_{i,v}$.

Finding the optimal multipliers is the main challenge in optimizing LD since it is nondifferentiable. There are several approaches; the most common is the subgradient method (see Shor et al. [21]). Starting from an arbitrary value of the multipliers u_0, it solves subproblems iteratively for different values of u. These are updated as follows:

$$u^{k+1} = u^k + t_k(y^k - x^k) \tag{9}$$

where the index k corresponds to the iteration number, y is a copy of variable x, and t_k is the step size. The step size is computed using the distance between the objective value of the preceding iteration, Z^{k-1}, and the estimated optimum Z^*:

$$t_k = \frac{\lambda(Z^{k-1} - Z^*)}{\|y^{k-1} - x^{k-1}\|^2} \qquad 0 \leq \lambda \leq 2. \tag{10}$$

Here, λ is a scaling factor used to control the convergence; it is normally between 0 and 2.

Our approach integrates an automatic bounding mechanism into the branch-and-bound algorithm of CP. The idea is that, at each node of the search tree, we use LD to yield valid bounds. The approach divides the problem into many subproblems; each subproblem has one global constraint and an objective function involving Lagrangian multipliers. For each subproblem, the solver must find the assignment that satisfies the global constraint while optimizing a linear objective function. It is generally possible to adapt the filtering algorithm of the global constraints to obtain an optimal support. If a constraint takes as parameter a cost for each pair for variable/value and it constrains the cumulative

cost of the assignments to be bounded, then this constraint is compatible with the bounding mechanism we propose. Soft constraints such as Cost-GCC and Cost-Regular [23] can therefore be used.

3.2 The Knapsack Constraint

For the MKP, the subproblems are 0/1 knapsack problems. They can be solved via dynamic programming, which runs in pseudo-polynomial time $\mathcal{O}(nW)$, where n is the number of items and W is the size of the knapsack. We use the algorithm by Trick [22] to filter the constraint, and we adapt it to compute the bound. Trick constructs an acyclic graph with $n+1$ layers where L_j contains the nodes of layer j for $0 \leq j \leq n$. Each node is labeled with a pair of integers: the layer and the accumulated weight. At layer 0, we have a single node $L_0 = \{(0,0)\}$. At layer j, we have $L_j = \{(j,k) \mid 1 \in \text{dom}(X_j) \wedge (j-1, k-w_j) \in L_{j-1} \vee 0 \in \text{dom}(X_j) \wedge (j-1,k) \in L_{j-1}\}$. There is an edge labeled with value 1 between the nodes $(j-1, k-w_j)$ and (j,k) whenever these nodes exist and an edge labeled with value 0 between the nodes $(j-1,k)$ and (j,k) whenever these nodes exist. Trick's algorithm filters the graph by removing all the nodes and edges that do not lie on a path connecting the node $(0,0)$ to a node (n,k) for $0 \leq k \leq C$. After the graph has been filtered, if no edges labeled with the value 1 remain between layer $j-1$ and layer j, this value is removed from the domain of x_j. Similarly, if no edges labeled with the value 0 remain between layer $j-1$ and layer j, this value is removed from the domain of x_j.

We augment this algorithm by computing costs. We assign to each edge connecting node $(j-1, k-w_j)$ to node (j,k) a cost of $c_j + \sum_{l=2}^{m} u(l,j)$ for the graph representing the first constraint ($i=1$), where c_j is the value of item j and m is the number of constraints; and a cost of $-u(i,j)$ where $i \geq 2$ is the index of the constraint for the graph representing the remaining constraints. We use dynamic programming to compute for each node (j,k) the cost of the longest path connecting the source node $(0,0)$ to the node (j,k):

$$M[j,k] = \begin{cases} 0 & \text{if } k = j = 0 \\ -\infty & \text{if } k > j = 0 \\ \max(M[j-1,k], M[j-1,k-w_j] + \mathit{addedCost}) & \text{otherwise} \end{cases}$$

where *addedCost* is equal to $c_j + \sum_{l=2}^{m} u(l,j)$ if $i=1$ and $-u(i,j)$ if $i>1$.

The optimal bound for this constraint is given by $\max_{0 \leq j \leq W} M[n,j]$. Indeed, any path in the filtered graph connecting the source node $(0,0)$ to a node (n,k) for $0 \leq k \leq C$ corresponds to a valid assignment. The expression $\max_{0 \leq j \leq W} M[n,j]$ returns the largest cost of such a path and therefore the maximum cost associated with a solution of the knapsack constraint.

3.3 The Regular Constraint

We proceed similarly with the regular constraint. Pesant [16] presents an algorithm also based on a layered graph. The set L_j contains the nodes at layer

j for $0 \leq j \leq n$. Each node is labeled with an integer representing the layer and a state of the automaton. The first layer contains a single node, and we have $L_0 = \{(0, q_0)\}$. The other layers, for $1 \leq j \leq n$, contain the nodes $L_j = \{(j, q_2) \mid (j-1, q_1) \in L_{j-1} \wedge a \in \mathrm{dom}(X_j) \wedge (q_1, a, q_2) \in \tau\}$. An edge is a triplet (n_1, n_2, a) where n_1 and n_2 are two nodes and $a \in A$ is a label. Two nodes can be connected to each other with multiple edges having distinct labels. The set of edges is denoted E. There is an edge connecting node $(j-1, q_1)$ to node (j, q_2) with label a whenever these nodes exist and $(q_1, a, q_2) \in \tau \wedge a \in \mathrm{dom}(X_j)$ holds. As with Trick's algorithm, Pesant filters the graph by removing all nodes and edges that do not lie on a path connecting the source node $(0, q_0)$ to a node (n, q) where $q \in F$ is a final state. If no edges with label a connecting a node in L_{j-1} to a node in L_j remain, the value a is pruned from the domain $\mathrm{dom}(X_j)$.

This filtering algorithm can be augmented by associating a cost with each edge. We assume that there is a Lagrangian multiplier $u(i, j, a)$ for each binary variable x_{ja} representing the assignment of variable X_j to character $a \in A$. Here, $i \geq 1$ represents the index of the constraint. An edge (q_1, a, q_2) has a cost of $p_{aj} + \sum_{l=2}^{m} u(l, j, a)$ in the graph induced by the first constraint ($i = 1$), and a cost of $-u(i, j, a)$ in the graph induced by the remaining constraints ($i > 1$). Using dynamic programming, we can compute the cost of the longest path connecting node $(0, q_0)$ to a node (j, q):

$$R[j, q] = \begin{cases} 0 & \text{if } j = 0 \\ \max_{((j-1,q_1),(j,q),a) \in E}(R[j-1, q_1] + addedCost) & \text{otherwise} \end{cases}$$

where $addedCost$ is equal to $p_{aj} + \sum_{l=2}^{m} u(l, j, a)$ if $i = 1$ and $-u(i, j, a)$ if $i > 1$.

Since every path from layer 0 to layer n in the graph corresponds to a valid assignment for the regular constraint, the optimal bound for this constraint is given by $\max_{q \in F} R[n, q]$, i.e., the greatest cost for a valid assignment.

3.4 The Subgradient Procedure

We use the subgradient procedure to solve the Lagrangian dual. The estimated optimum Z^* is set to the value of the best incumbent solution, and the scaling factor λ is set to 2. The updating strategy halves λ when the objective function does not improve in five iterations. At each node of the search tree, the number of iterations for the subgradient procedure, which gives the best trade-off between the running time and the number of nodes required in the search tree, is fixed to 60 for the MKP in which the subproblem is solved by dynamic programming and to 10 for all other cases. The subgradient procedure is terminated as soon as the resulting Lagrangian bound is inferior to the value of the best incumbent solution.

The initial multipliers u_0 at the root node are fixed to 1. At the other nodes, we use an inheritance mechanism, i.e., the value of u_0 at a node is taken from the multipliers of the parent node. This mechanism is better than always fixing the initial multipliers to a given value. We have tested two strategies: we fix u_0 either to the multipliers that give the best bound or to the last multipliers of the

parent node. The results show that the latter strategy generally performs better. This is because the limited number of subgradient iterations at each node is not sufficient to provide tight bounds for the overall problem. Therefore, the more iterations performed at early nodes, the better the multipliers expected at later nodes.

4 Computational Results

This section reports some experimental results. The goal is not to present state-of-the-art results for specific problems but to show that LD could make CP tools more efficient for optimization problems. The algorithms are built around CP Optimizer 12.6 with depth-first search. All the other parameters of the solver are set to their default values. The criteria used to evaluate the solutions are the number of nodes in the search tree, the computational time, and the number of instances successfully solved to optimality.

We investigate the behavior of the algorithms in two contexts: with and without initial lower bounds. In case A, we provide the optimal solution to the model by adding a constraint on the objective function, setting it to be at least the value of the optimal solution; the goal is then to prove the optimality of this value. In case B, we start the algorithms without an initial lower bound. These two tests provide different insights, since the presence of different bounding mechanisms will impact the search and branching decisions of the solver, with unpredictable outcomes. However, once the solver has found the optimal solution, it must always perform a last search to demonstrate optimality. A better bounding mechanism will always contribute to this phase of the overall search.

4.1 Results for MKP

The experiments in this subsection analyze the performance of LD with two MKP solution methods: i) direct CP using only sum constraints, and ii) the approach of Trick [22]. For i) the Lagrangian subproblems are solved by dynamic programming, and for ii) they are solved using the filtering graph of Trick [22].

We selected two groups of small and medium MKP instances from the OR-Library [1] for the test: Weing (8 problems, each with 2 constraints) and Weish (30 problems, each with 5 constraints). We limited the computational time to one hour.

Table 1 reports the results for case B and Table 2 reports the results for case A. The instances not included in Table 1 could not be solved by any of the algorithms; a dash indicates that the algorithm could not solve the instance within the time limit.

As can be seen in Table 2, the LD approach processes much fewer nodes per seconds, but the extra effort invested in bounding significantly decreases the size of the search tree and the time to find the optimal solution, especially for the larger instances. The most efficient method is CP + LD: it can solve 30 instances. However, it generally requires more time to solve the small instances. Moreover,

it is quite time-consuming compared with the original method on a few instances, e.g., Weish6, Weish7, Weish8, and Weing15. The Trick + LD approach improves on Trick for all the criteria, even for the small instances. This is because solving the Lagrangian subproblems based on the information available from the filtering algorithm is computationally less costly. Moreover, the propagation performed at each node of the search tree is limited for the direct CP formulation, and the computation of the Lagrangian bound significantly increases this. On the other hand, the Trick method relies on a filtering algorithm for which the complexity is linear in the size of the underlying graph. In this case, the additional computation for the LD bounds has less impact.

Instance	#	CP			CP + LD			Trick			Trick + LD		
	Vars	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$
Weing1	28	4424	1.19	3718	860	3.95	218	4424	6.19	715	1100	6.00	183
Weing2	28	5572	1.29	4319	744	3.07	242	5572	11.94	467	744	4.30	173
Weing3	28	8650	0.55	16k	270	0.97	278	8650	15.34	564	280	1.53	183
Weing4	28	4106	1.08	3802	538	2.62	205	4106	19.72	208	538	4.14	130
Weing5	28	13k	0.58	22k	262	1.00	262	12k	21.13	615	262	1.53	171
Weing6	28	9150	1.14	8026	876	3.59	244	9150	20.50	446	1012	4.83	210
Weing7	105	-	-	-	32k	3410.04	9	-	-	-	-	-	-
Weing8	105	-	-	-	19k	147.98	128	-	-	-	19k	655.46	29
Weish1	30	35k	1.78	20k	1320	37.31	35	35k	910.82	38	1286	59.90	21
Weish2	30	40k	1.64	24k	1280	27.14	47	40k	2481.72	16	1384	57.47	24
Weish3	30	11k	0.83	13k	674	8.47	80	11k	669.01	16	760	24.35	31
Weish4	30	2342	0.80	2927	856	11.91	72	2342	186.47	13	826	29.68	28
Weish5	30	1614	0.90	1793	728	9.19	79	1614	149.78	11	644	23.74	27
Weish6	40	1.2M	12.56	96k	4286	369.13	12	-	-	-	3368	320.92	10
Weish7	40	901k	15.12	60k	3888	290.79	13	-	-	-	3482	352.67	10
Weish8	40	1.1M	19.74	56k	4004	256.00	16	-	-	-	3464	392.57	9
Weish9	40	144k	3.12	46k	2426	46.78	52	-	-	-	2212	191.83	12
Weish10	50	28M	589.20	48k	4486	264.40	17	-	-	-	3969	734.65	5
Weish11	50	6M	95.84	63k	3764	159.05	24	-	-	-	3764	595.38	6
Weish12	50	21M	355.78	59k	4738	307.37	15	-	-	-	3728	651.90	6
Weish13	50	20M	338.07	59k	4208	250.62	17	-	-	-	4374	908.53	5
Weish14	60	-	-	-	8424	645.04	13	-	-	-	8856	2472.92	4
Weish15	60	35M	720.98	49k	13k	1363.17	10	-	-	-	12k	3482.11	3
Weish16	60	-	-	-	14k	1216.45	12	-	-	-	13k	3324.81	4
Weish17	60	-	-	-	14k	1600.78	9	-	-	-	-	-	-
Weish18	70	-	-	-	-	-	-	-	-	-	-	-	-
Weish19	70	-	-	-	7750	667.02	12	-	-	-	8907	3174.07	3
Weish20	70	-	-	-	18k	2610.74	7	-	-	-	-	-	-
Weish21	70	-	-	-	15k	1642.16	9	-	-	-	-	-	-
Weish22	80	-	-	-	15k	2905.41	5	-	-	-	-	-	-
Weish23	80	-	-	-	12k	2002.74	6	-	-	-	-	-	-

Table 1. Results for MKP without initial lower bound

Table 2 presents the results for case A. Here, the Lagrangian approaches are even more efficient. The algorithms with LD can successfully solve all 38 instances, and the search tree is relatively small. The computational time of CP

+ LD is still worse than that of CP on a few small instances, but the difference has decreased significantly. The main reason for this is that the optimal solutions used to compute the step size make the subgradient procedure more stable and cause it to converge more quickly. This observation suggests that adding good lower bounds as soon as possible in the solution process will improve the method.

We computed at the root of the search tree the bound that the LD provides as well as the bound returned by CPLEX using a linear relaxation (LP) and the bound that CPLEX computes when the integer variables are not relaxed and all cuts are added (ILP). Due to lack of space, we do not report the bounds for

Instance	#	CP			CP + LD			Trick			Trick + LD		
	Vars	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$
Weing1	28	3408	0.21	16k	24	0.62	39	3408	5.25	649	24	0.78	31
Weing2	28	5070	0.21	24k	30	0.48	63	5070	16.36	310	30	0.74	41
Weing3	28	7004	0.55	13k	32	0.32	100	7004	11.78	595	34	0.42	81
Weing4	28	2344	0.14	17k	16	0.37	43	2344	10.45	224	16	0.81	20
Weing5	28	18k	0.33	55k	26	0.32	81	18k	35.30	510	26	0.56	46
Weing6	28	8038	0.22	37k	30	0.44	68	8038	27.62	291	30	0.80	38
Weing7	105	-	-	-	134	4		-	-	-	134	174.03	1
Weing8	105	-	-	-	168	3.54	47	-	-	-	520	40.95	13
Weish1	30	11k	0.27	41k	54	2.85	19	11k	447.18	25	70	5.72	12
Weish2	30	23k	0.43	53k	66	1.98	33	23k	1679.54	14	62	9.15	7
Weish3	30	8394	0.23	36k	38	2.80	14	8394	476.57	18	46	5.59	8
Weish4	30	632	0.14	4514	34	1.26	27	632	57.19	11	38	4.64	8
Weish5	30	556	0.12	4633	34	0.87	39	556	55.93	10	36	5.67	6
Weish6	40	861k	12.92	67k	56	5.54	10	-	-	-	72	16.58	4
Weish7	40	456k	7.07	64k	60	8.40	7	-	-	-	72	17.40	4
Weish8	40	707k	10.38	68k	60	9.05	7	-	-	-	54	19.52	3
Weish9	40	74k	1.26	59k	48	2.28	21	-	-	-	74	18.18	4
Weish10	50	8.6M	132.92	65k	134	29.53	5	-	-	-	192	51.11	4
Weish11	50	1.4M	22.75	62k	86	10.50	8	-	-	-	82	28.44	3
Weish12	50	9M	135.70	66k	114	24.92	5	-	-	-	122	38.23	3
Weish13	50	4.1M	60.90	67k	120	17.84	7	-	-	-	112	35.23	3
Weish14	60	-	-	-	104	27.97	4	-	-	-	330	123.89	3
Weish15	60	9.7M	173.65	56k	92	28.20	3	-	-	-	146	68.70	2
Weish16	60	-	-	-	128	37.95	3	-	-	-	450	192.18	2
Weish17	60	156M	3537.25	44k	90	34.36	3	-	-	-	176	115.41	1.5
Weish18	70	-	-	-	200	87.97	2	-	-	-	142	116.60	1.2
Weish19	70	-	-	-	200	86.98	2	-	-	-	320	228.97	1.4
Weish20	70	-	-	-	174	65.84	3	-	-	-	596	340.87	1.7
Weish21	70	-	-	-	134	48.17	3	-	-	-	354	225.53	1.6
Weish22	80	-	-	-	150	87.39	2	-	-	-	1068	1083.83	1.0
Weish23	80	-	-	-	158	83.39	1.9	-	-	-	177	350.55	0.5
Weish24	80	-	-	-	146	84.90	1.7	-	-	-	154	194.55	0.8
Weish25	80	-	-	-	238	117.47	2	-	-	-	586	539.24	1.1
Weish26	90	-	-	-	178	135.75	1.3	-	-	-	438	567.02	0.8
Weish27	90	-	-	-	152	94.48	1.6	-	-	-	258	356.33	0.7
Weish28	90	-	-	-	152	96.71	1.6	-	-	-	266	344.16	0.8
Weish29	90	-	-	-	152	104.87	1.4	-	-	-	416	601.38	0.7
Weish30	90	-	-	-	152	126.96	1.2	-	-	-	138	372.65	0.4

Table 2. Results for MKP: proving optimality

each instance. In average, the bounds for LD, LP, and ILP are 1.03%, 0,79%, and 0.22% greater than the optimal value. Even though LD does not provide the best bound, it represents a significative improvement to CP whose bound is very weak.

4.2 Results for SSP

To generate the regular constraints, we use the procedure proposed by Pesant [16] to obtain random automata. The proportion of undefined transitions in τ is set to 30% and the proportion of final states to 50%. To control the problem size, we create instances with only 2 regular constraints and 50 periods. The assignment profits are selected at random from integer values between 1 and 100. Each instance is then generated based on two parameters: the number of activities (nva) and the number of states (nbs) in the automata. For each pair (nva,nbs), we randomly generate three instances. The instances are labeled nbv-nbs-i, where i is the instance number. Because SSP instances are more difficult than MKP instances, the computational time is increased to 2 hours.

The experiment investigates the performance of LD when combined with the method of Pesant [16] for the SSP. As for the MKP, we test two cases: case A has initial lower bounds and case B does not. Tables 3 and 4 present the results, which demonstrate the performance of our approach. The LD method clearly improves the method of Pesant [16] for all three criteria. More precisely, for case B, it can solve 6 additional instances, reducing the size of the search tree by a factor of about 7 and the computational time by a factor of 1.5 on average. In case A, it works even better, reducing the size of the search tree by a factor of about 16 and the computational time by a factor of more than 4 on average. This is because we can reuse the graph constructed by the filtering algorithm to solve the Lagrangian subproblems.

Instance	Pesant			Pesant + LD		
	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$
10-20-01	2.2M	1232.72	1785	156k	346.92	450
10-20-02	-	-	-	298k	741.35	402
10-20-03	1.6M	783.29	2043	158k	332.15	476
10-80-01	256k	1325.80	193	84k	1020.53	82
10-80-02	788k	3307.39	238	238k	2834.86	84
10-80-03	847k	2344.55	361	246k	2176.73	113
20-20-01	-	-	-	828k	1856.10	446
20-20-02	-	-	-	1.3M	3404.56	382
20-20-03	2.1M	2427.72	865	164k	439.97	373
20-80-01	-	-	-	373k	3944.18	95
20-80-02	-	-	-	436k	5206.81	84
20-80-03	-	-	-	228k	2561.64	89

Table 3. Results for SSP without initial lower bound

Instance	Pesant			Pesant + LD		
	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$	Nodes	Time	$\frac{\text{Nodes}}{\text{Time}}$
10-20-01	87k	42.20	2062	2564	5.94	432
10-20-02	407k	342.43	1189	44k	108.30	406
10-20-03	71k	28.60	2483	4118	7.48	551
10-80-01	20k	118.97	168	4546	71.16	64
10-80-02	25k	81.39	307	5466	63.19	87
10-80-03	26k	85.67	303	4762	58.34	82
20-20-01	343k	176.38	1945	2651	13.41	198
20-20-02	372k	297.97	1248	10k	51.63	194
20-20-03	53k	44.28	1197	1353	7.43	182
20-80-01	105k	486.89	216	10k	147.63	68
20-80-02	216k	1648.28	131	13k	211.23	62
20-80-03	16k	128.74	124	5128	72.33	71

Table 4. Results for SSP: proving optimality

5 Conclusions and future work

We have introduced an automatic bounding mechanism based on the LD concept to improve the performance of CP for optimization problems. We tested the approach on two problems, the MKP and a synthesized version of the SSP. These rely on two well-known global constraints: knapsack and regular.

Our next step will be to apply the approach to other global constraints to solve real problems. One candidate is the global cardinality constraint, for which the subproblems can be solved in polynomial time with the aid of a graph obtained from the filtering algorithm. In addition, our approach could be improved in several ways. First, we could use the information obtained from solving the Lagrangian dual to design a strong cost-based filtering, as proposed by Cronholm and Ajili [5]. Second, by checking the feasibility of the solutions of the Lagrangian subproblems we could find good solutions that help to limit the search and improve the convergence of the subgradient method. Moreover, we could use the solutions of the subproblems to guide the search by branching on the variable with the largest difference between itself and its copies. Finally, as reported by Guignard [9], the subgradient approach can have unpredictable convergence behavior, so a more suitable algorithm could improve the performance of our approach.

Acknowledgment

This work was financed with a Google Research Award. We would like to thank Laurent Perron for his support.

References

[1] J.E Beasley. *OR-library*, 2012. http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

[2] Thierry Benoist, Franois Laburthe, and Benot Rottembourg. Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. In *CP-AI-OR'2001, Wye College*, pages 15–26, 2001.

[3] Nicolas Chapados, Marc Joliveau, Pierre L'Ecuyer, and Louis-Martin Rousseau. Retail store scheduling for profit. *European Journal of Operational Research*, 239(3):609–624, 2014. doi: 10.1016/j.ejor.2014.05.033. URL http://dx.doi.org/10.1016/j.ejor.2014.05.033.

[4] Marie-Claude Côté, Bernard Gendron, Claude-Guy Quimper, and Louis-Martin Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 16(1):54–76, 2011. doi: 10.1007/s10601-009-9083-2. URL http://dx.doi.org/10.1007/s10601-009-9083-2.

[5] Wilhelm Cronholm and Farid Ajili. Strong cost-based filtering for Lagrange decomposition applied to network design. In *Principles and Practice of Constraint Programming CP 2004*, pages 726–730, 2004. URL http://www.springerlink.com/index/ur3uvyqbp0216btd.pdf.

[6] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006. URL http://dblp.uni-trier.de/db/journals/constraints/constraints11.html#DemasseyPR06.

[7] Torsten Fahle and Meinolf Sellmann. Cost based filtering for the constrained knapsack problem. *Annals of OR*, 115(1–4):73–93, 2002. doi: 10.1023/A:1021193019522. URL http://dx.doi.org/10.1023/A:1021193019522.

[8] Daniel Fontaine, Laurent D. Michel, and Pascal Van Hentenryck. Constraint-based Lagrangian relaxation. In *CP'14*, pages 324–339, 2014.

[9] Monique Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003. ISSN 1134-5764. doi: 10.1007/BF02579036. URL http://dx.doi.org/10.1007/BF02579036.

[10] Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39(2):215–228, 1987. ISSN 00255610. doi: 10.1007/BF02592954.

[11] Irit Katriel, Meinolf Sellmann, Eli Upfal, and Pascal Van Hentenryck. Propagating knapsack constraints in sublinear time. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*, pages 231–236, 2007. URL http://www.aaai.org/Library/AAAI/2007/aaai07-035.php.

[12] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[13] Yuri Malitsky, Meinolf Sellmann, and Radoslaw Szymanek. Filtering bounded knapsack constraints in expected sublinear time. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010*, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1855.

[14] Julien Menana and Sophie Demassey. Sequencing and counting with the multicost-regular constraint. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5547 LNCS, pages 178–192, 2009. ISBN 3642019285. doi: 10.1007/978-3-642-01929-6_14.

[15] Wided Ouaja and Barry Richards. A hybrid multicommodity routing algorithm for traffic engineering. *Networks*, 43(3):125–140, 2004. URL `http://dblp.uni-trier.de/db/journals/networks/networks43.html#OuajaR04`.

[16] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *Principles and Practice of Constraint Programming CP 2004*, pages 482–495, 2004. URL `http://www.springerlink.com/content/ed24kyhg561jjthj`.

[17] Claude-Guy Quimper and Louis-Martin Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *J. Heuristics*, 16(3):373–392, 2010. doi: 10.1007/s10732-009-9106-6. URL `http://dx.doi.org/10.1007/s10732-009-9106-6`.

[18] Meinolf Sellmann. Approximated consistency for knapsack constraints. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, pages 679–693, 2003. doi: 10.1007/978-3-540-45193-8_46. URL `http://dx.doi.org/10.1007/978-3-540-45193-8_46`.

[19] Meinolf Sellmann. Theoretical foundations of CP-based Lagrangian relaxation. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 634–647. Springer, 2004. ISBN 3-540-23241-9. URL `http://dblp.uni-trier.de/db/conf/cp/cp2004.html#Sellmann04`.

[20] Meinolf Sellmann and Torsten Fahle. Constraint programming based Lagrangian relaxation for the automatic recording problem. In *Annals of Operations Research*, volume 118, pages 17–33, 2003. ISBN 0254-5330. doi: 10.1023/A:1021845304798.

[21] N. Z. Shor, Krzysztof C. Kiwiel, and Andrzej Ruszcayǹski. *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-12763-1.

[22] Michael A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of OR*, 118(1–4):73–84, 2003. URL `http://dblp.uni-trier.de/db/journals/anor/anor118.html#Trick03`.

[23] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *J. Heuristics*, 12(4–5): 347–373, 2006. doi: 10.1007/s10732-006-6550-4. URL `http://dx.doi.org/10.1007/s10732-006-6550-4`.