Master Operations Research and Combinatorial Optimization
Master Informatique / Master Mathématiques & Applications

Lower Bounds Computation on Collections Based Models

Benjamin Collet

June 28, 2019

Research project performed at

Under the supervision of:
Julien Darlay

Defended before a jury composed of:
Prof Van-Dat Cung
Dr Matěj Stehlík
Prof Nadia Brauner
Dr Julien Darlay

June 2019

Abstract

Optimization software, in an industrial context, are often required to run in very short time. Local search heuristics are suited for this kind of requirement, but do not provide proof of optimality by themselves. This work describe the column generation method, and how it has been used to quickly generate bounds on problems like bin packing and vehicles routing. Difficulties encountered during the implementation are also discussed, in particular the solving of the slave problems used to generate the columns. The resulting bounds are not perfect but can be obtained in a short amount of time, and thus are very scalable. The prototype developed being a successful proof of concept, column generation will likely be implemented in LocalSolver.

Résumé

Dans un contexte industriel, les logiciels d'optimisation doivent souvent fournir des résultats très rapidement. Les heuristiques basées sur la recherche locale sont particulièrement adaptées à ce genre d'utilisation, mais sont incapables de fournir une preuve d'optimalité. Ce document décrit la méthode de génération de colonnes, et comment elle peut être utilisée pour calculer rapidement des bornes sur des problèmes de bin packing et de tournées de véhicules. Les difficultés rencontrées lors de du développement y sont aussi rapportées, en particulier la résolution des problèmes esclaves utilisés pour générer les colonnes. Les bornes obtenues ne sont pas parfaites mais peuvent être calculées en un temps très court, ce qui permet un bon passage à l'échelle. Le prototype réalisé ayant démontré l'efficacité de la génération de colonnes, cette méthode sera implémentée dans LocalSolver.

Contents

— 1 —
Introduction

Optimization software, in an industrial context, are often required to run in very short time (few seconds). This can be a problem to apply exact methods, as instances are usually quite large. But often, good results are enough and optimality is not required. In this context, heuristic methods such as local search are a good solution as they can rapidly yield good results, even if they are unable to prove optimality or even an optimality gap.

Located in Paris, and part of the french industrial group Bouygues, LocalSolver is a small company of 14 people. Its main activities are to develop and sell the optimisation solver of the same name, and to carry out custom software projects for the industry, usually using the aforementioned solver as the optimisation engine. Thus, the solver is made to be used by the industry and aims at being generic, efficient and scalable. It has its own formalism which is design to easily describe almost any mathematical program. The solver is mainly based on local search, but also as a mixed integer programming (MIP) solver and a nonlinear programming (NLP) solver integrated, as they might be more efficient on some problems.

Local search does not provide proof of optimality by itself and formal methods like MIP and NLP do not scale well. But even when they are not needed, bounds are still valuable, at least to estimate the quality of a provided solution. It is within this context that I have realized a prototype that provide bounds on the bin packing problem (BPP) and the capacitated vehicle routing problem (CVRP) using the column generation method, as a proof of concept to know if it is worth implementing something similar in the solver.

First, we will see in more details the context and the problem studied during this research project. Then we will describe the state of the art and explain the theory behind the chosen solution. Implementation details and some encountered difficulties will be discussed, as well as an analysis of the obtained results.

— 2 —
Analysis of the problem

2.1 Context of optimization softwares

LocalSolver is mainly based on local search as it aims to be efficient and scalable. This is looked for by the industry, as they have large scale problems that need good solutions in short amount of time. Good solutions that can be obtained quickly are sometime more valuable than optimal solutions, and local search heuristics seems well adapted in these cases.

In industrial application, local search methods uses a large neighbourhood to be able to converge quickly, and to minimize the odds of getting stuck in a local minimum. But this large neighbourhood can cause the solver to take a very long time to stop and it can be beneficial to know early that an optimal solution have been reach to stop the search.

Even if the industry does not always needs proof of optimality (or optimality gap), it is sometime useful to know the quality of the implemented solution. During the development phase of big projects, it is common to run dedicated software to be sure that the developed solution is close enough to the optimality. For example, during a project about xDSL deployment for Bouygues Telecom, a commercial MIP solver was run on a relaxation during a night to get a bound and compute a gap, but this is not possible during production where run-times are expected to be much shorter (less than a minute). To be more precise, this approach is not scalable.

The original idea behind LocalSolver is to provide a generic engine, based on local search, that is able to solve any kind of problem. Being able to get useful bounds in the same generic manner as the optimization engine is not an easy task. A good part of this job is done by the MIP solver presents inside LocalSolver. In addition to the MIP, there is also a NLP solver that can handle trickier models. The model is parsed into an expression tree (DAG) that is used by all components, and for a given model, some components will be quicker than others. Also some components can only give feasible solutions and other can will yield bounds. When everything is combined we obtain a very powerful tool.

In addition to the classical Boolean, integer and floating-point variable types, LocalSolver also provide variable types called set and list that can be considered higher level[1]. Set and list are defined on an interval $[0, n-1]$ and contain unique elements (positive integer) of that interval, in an ordered way for lists. More formally, sets are subset of the defining interval and list are permutation of a subset of this interval. Note that because of this definition, lists (like sets)

[1] `https://www.localsolver.com/docs/8_5/advancedfeatures/collectionvariables.html`

cannot contain repeated element. To make the best use of these collections, specific operators have been introduced:

- `count` returns the number of elements in a collection;
- `contains` returns a Boolean corresponding to the presence of an element.

Some operators are made for an arbitrary number of collections defined on the same domain:

- `disjoint` returns true when all collections are pairwise disjoint, that is no value appears in more than one collection (collections need to be defined on the same domain);
- `partition` returns true when the given collections form a partition of their definition set. That is all value are present exactly once.

And finally, some operator are specific to lists:

- `at` access the value at a given position in the list (can be used to constrain relative positioning, like an element that need to appear before another);
- `indexOf` returns the position of a given integer in the list.

We now face the problem of providing bounds on models that constrains these variables.

2.2 Case study

To illustrate these new type of variable, we will detail the modeling of two problem: BPP and CVRP. They both have an simple reformulation using respectively sets and lists variables types.

2.2.1 Bin packing problem

The first problem to be studied is the bin packing. Let C the capacity of a bin and w_i the weight of item i. Our decision variables are x_{ij}, set to 1 if item i is placed in bin j and 0 otherwise, and y_j set to 1 if bin j is used and 0 otherwise.

$$\begin{aligned} \text{minimize} \quad & \sum_j y_j \\ \text{subject to} \quad & \sum_j x_{ij} = 1 \qquad \forall i \in I, \\ & \sum_i x_{ij} w_i \leq y_j C \quad \forall j \in J, \\ & x_{ij}, y_j \in \{0,1\} \end{aligned} \tag{2.1}$$

To reformulate the model of the bin packing problem by making use of the set variable type, a bin can be represented as a set. In (2.1), the first constraint express that all items are placed in exactly one box. This constraint is equivalent to the `partition` constraint introduced with the set type. Having that constraint built-in mean that the internal model is aware that these are only one constraint and can use specific moves. Here is what the problem look like written in LSP language[2]:

[2] `https://www.localsolver.com/docs/8_5/exampletour/binpacking.html`

```
/* Declares the optimization model. */
function model() {
    // Set decisions: bins[k] represents the items in bin k
    bins[k in 0..nbMaxBins-1] <- set(nbItems);

    // Each item must be in one bin and one bin only
    constraint partition[k in 0..nbMaxBins-1](bins[k]);

    for [k in 0..nbMaxBins-1] {
        // Weight constraint for each bin
        binWeights[k] <- sum(bins[k], i => itemWeights[i]);
        constraint binWeights[k] <= binCapacity;

        // Bin k is used if at least one item is in it
        binsUsed[k] <- (count(bins[k]) > 0);
    }

    // Count the used bins
    totalBinsUsed <- sum[k in 0..nbMaxBins-1](binsUsed[k]);

    // Minimize the number of used bins
    minimize totalBinsUsed;
}
```

Note that this model, while using more complex variables, only have $O(n)$ of them, instead of $O(n^2)$ variables in (2.1). This gap will usually be even larger on models that use lists like vehicle routing problems (VRPs).

2.2.2 Vehicle routing problem

To illustrate the list variable type, one of the most natural problem is the VRP. In the case of the capacitated version, there is clear similarity with the bin packing. The main difference is that vehicle visit each of their customer in a specific order, represented by the list order. The resulting LSP model[3] is quite similar to the bin packing:

```
/* Declares the optimization model. */
function model() {
    // Sequence of customers visited by each truck.
    customersSeq[k in 1..nbTrucks] <- list(nbCustomers);

    // All customers must be visited by  the trucks
    constraint partition[k in 1..nbTrucks](customersSeq[k]);

    for [k in 1..nbTrucks] {
        local seq<- customersSeq[k];
        local c <- count(seq);
```

[3] https://www.localsolver.com/docs/8_5/exampletour/vrp.html

```
        // A truck is used if it visits at least one customer
        trucksUsed[k] <- c > 0;

        // The quantity needed in each route must not exceed the truck capacity
        routeQuantity <- sum(0..c-1, i => demands[seq[i]]);
        constraint routeQuantity <= truckCapacity;

        // Distance travelled by truck k
        routeDist[k] <- sum(1..c-1, i => distMatrix[seq[i - 1]][seq[i]])
             + (c > 0 ? (distWarehouse[seq[0]] + distWarehouse[seq[c - 1]]) : 0);
    }

    nbTrucksUsed <- sum[k in 1..nbTrucks](trucksUsed[k]);

    // Total distance travelled
    totalDist <- sum[k in 1..nbTrucks](routeDist[k]);

    // Objective: minimize the distance travelled
    minimize totalDist;
}
```

The main differences are that the computing of the capacity and the traveled distance is more complicated than for the bin packing.

In the next sections we will described generic techniques to compute lower bounds on problems that uses sets and lists variables.

— 3 —
State of the art

Recall that in LocalSolver, the collections are defined on a domain. Let I be the set of elements of that domain. In a BPP, I is the set of item to pack, in a VRP, I will be the set of customers to visit.

Collections will usually have some constraints, like a weight limit on the bins in a BPP. Let J be the set of all valid collections in our problem (e.g. for the BPP, the set of all possible way to fill a bin).

Let c_j be the cost of the collection j and x_j a decision variable set to 1 if j appear in the solution, 0 otherwise.

Most of the collections problems with partition (or disjoint) constraints may be formulated as the following MIP (replacing $= 1$ by ≤ 1 for disjoint). The objective is to minimize the total cost such that each item appears in one selected element of J.

$$\begin{aligned} \text{minimize} \quad & \sum_j c_j x_j \\ \text{subject to} \quad & \sum_{j \in J : j \supset i} x_j = 1 \qquad \forall i \in I, \\ & x_j \in \{0,1\} \quad \forall j \in J \end{aligned} \tag{3.1}$$

Example

We will follow a trivial example of BPP, with tree items of weight 1, and a maximum bin capacity of 2. Let $I = \{A,B,C\}$ and $J = \{\emptyset, \{A\}, \{B\}, \{C\}, \{A,B\}, \{A,C\}, \{B,C\}\}$.

$$\begin{aligned} \text{minimize} \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\ \text{subject to} \quad & x_2 + x_5 + x_6 = 1, \\ & x_3 + x_5 + x_7 = 1, \\ & x_4 + x_6 + x_7 = 1, \\ & x_j \in \{0,1\} \quad \forall j \in [1,7] \end{aligned}$$

Note that in a BPP, each bin cost 1, and thus c_j don't appears.

As we will see, this problem can be huge, and we are only looking for bounds, so we can relax the integrity constraint to $x_j \in [0,1]$. The first constraint already enforce $x_j \leq 1$, so our

relaxed model can be written as such:

$$\begin{aligned} z^\star = \text{minimize} \quad & \sum_j c_j x_j \\ \text{subject to} \quad & \sum_{j \in J : j \supset i} x_j = 1 \quad \forall i \in I, \\ & x_j \geq 0 \quad \forall j \in J \end{aligned} \tag{3.2}$$

Stated this way, the number of columns is exponential and the model is too big for real world application. For a typical BPP, $|J| = \mathrm{O}(2^n)$, and can be even bigger for VRP or other lists based problem.

Problem (3.2)can be represented in a more compact way when considering only a feasible solution. Let B the base of a basic solution of (3.2), let J_B the set of collection in B, and $J_{\overline{B}}$ the set of collection not in B, i.e. $J_{\overline{B}} = \{j : x_j = 0 \text{ in the primal solution}\}$:

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in J_B} c_j x_j + \overbrace{\sum_{j \in J_{\overline{B}}} c_j x_j}^{=0} \\ \text{subject to} \quad & \sum_{j \in J_B : j \supset i} x_j + \underbrace{\sum_{j \in J_{\overline{B}} : j \supset i} x_j}_{=0} = 1 \quad \forall i \in I, \\ & x_j \geq 0 \quad \forall j \in J_B, \\ & x_j = 0 \quad \forall j \in J_{\overline{B}} \end{aligned}$$

And removing the parts equal to 0:

$$\begin{aligned} z = \text{minimize} \quad & \sum_{j \in J_B} c_j x_j \\ \text{subject to} \quad & \sum_{j \in J_B : j \supset i} x_j = 1 \quad \forall i \in I, \\ & x_j \geq 0 \quad \forall j \in J_B \end{aligned} \tag{3.3}$$

Example

Applying that to our example, with 2, 3, and 4 in base:

$$\begin{aligned} \text{minimize} \quad & x_2 + x_3 + x_4 + \overbrace{x_1 + x_5 + x_6 + x_7}^{=0} \\ \text{subject to} \quad & x_2 + x_5 + x_6 = 1, \\ & x_3 + x_5 + x_7 = 1, \\ & x_4 + \underbrace{x_6 + x_7}_{=0} = 1, \\ & x_j \geq 0 \quad \forall j \in [2,3,4], \\ & x_j = 0 \quad \forall j \notin [2,3,4] \end{aligned}$$

Simplified in:

$$\begin{aligned} \text{minimize} \quad & x_2 + x_3 + x_4 \\ \text{subject to} \quad & x_2 = 1, \\ & x_3 = 1, \\ & x_4 = 1, \\ & x_j \geq 0 \quad \forall j \in [2,3,4] \end{aligned}$$

This is linear in size, as $|J_B| = \mathrm{O}(n)$, but only capture a small part of the original problem. Let y_B the dual values, and r_j the reduce cost associated with to a column $A_j \in J$. We have:

$$\begin{aligned} x_{J_B} &= B^{-1}\mathbb{1} \\ x_{J_{\overline{B}}} &= 0 \\ y_B &= c_B B^{-1} \\ r_j &= c_j - y_B A_j \end{aligned}$$

Example

For our example, $B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$$\begin{aligned} x_{2,3,4} &= 1 \\ x_{1,5,6,7} &= 0 \\ y_B &= \{1,1,1\} \\ r_{[1,4]} &= 0 \\ r_{[5,7]} &= -1 \end{aligned}$$

B can be obtain by multiple ways, in our case, we will take a feasible solution from LocalSolver. By construction, B is the optimal solution for (3.3). We want to know if B is the optimal solution of (3.2). A classical result in linear programming (LP) [4] gives us the following property:

$$\begin{aligned} B \text{ opt} &\iff r_j \geq 0 \quad \forall j \in J_{\overline{B}} \\ &\iff r_j \geq 0 \quad \forall j \in J \end{aligned}$$

$J_{\overline{B}}$ and J are still big, but we can just find the minimum reduced cost, and if it is greater than 0, then B is optimal for (3.2).

$$\min_{j \in J} r_j = \min_{j \in J} \left(c_j - y_B A_j\right)$$

This problem is called the slave problem, and can be solved as a MIP or on a case-by-case basis with dedicated algorithms.

Let $A_j = \begin{pmatrix} z_1 \\ \vdots \\ z_i \\ \vdots \\ z_B \end{pmatrix}$ be a column of J described by the Boolean decisions variables z_i. The computation of the minimal reduced cost is equivalent to the solution of the following problem:

$$\begin{aligned} r^\star = \underset{z}{\text{minimize}} \quad & \sum_i c_{A_j} - y_{B_i} z_i \\ \text{subject to} \quad & \begin{pmatrix} \vdots \\ z_i \\ \vdots \end{pmatrix} \in J \end{aligned} \tag{3.4}$$

Example

Let $A_j = \begin{pmatrix} z_A \\ z_B \\ z_C \end{pmatrix}$. As this is a BPP, $c_{A_j} = 1$, so we can write the slave problem of our example as:

$$\begin{aligned} \text{minimize} \quad & 1 - z_A + z_B + z_C \\ \text{subject to} \quad & z_A + z_B + z_C \leq 2 \end{aligned}$$

That can be rewritten as:

$$\begin{aligned} 1 - \text{maximize} \quad & z_A + z_B + z_C \\ \text{subject to} \quad & z_A + z_B + z_C \leq 2 \end{aligned}$$

Solving the slave problem (3.4) give A_j, a "column" of minimum reduced cost $r^\star$. As stated before, if $r^\star$ is strictly positive then the current solution is optimal for (3.2). Else, we can addA_j inside (3.3) and solve it again, and repeat until optimality is proven. Dantzig-Wolfe decomposition [5] generate formulation like (3.1). Linear relaxation of these problems (i.e. (3.2)) can be solved by the process described earlier, known as column generation [1, 4, 7, 10, 13]. A method called branch and price [13, 15] can then be used to solve the original problem, or tighten the bound.

While the whole process of the column generation can give us an optimal solution of (3.2), we can compute a lower bound at any step [7, 13]. Let z the solution of (3.3), the current master, and $z^\star$ the optimal solution of (3.2). Let κ be an upper bound on $\sum_{j \in J} x_j^\star$, with $x_j^\star$ being the optimal solution of (3.2). We have:

$$z + r^\star \kappa \leq z^\star \leq z \tag{3.5}$$

The intuition is that we can replace at most each variable in base, and that each replacement can improve the solution by at most the minimum reduced cost.

— 4 —
Implementation

The development happened outside of LocalSolver, only using it as a library. Even if we want to integrate column generation bounds to the solver, this project was just a prototype. Since our MIP solver only have C++ and C# interfaces, only these two languages where available. We choose C#, as it would have slowed the development to do it in C++ and a performance gain that was not needed.

4.1 Bin packing problem

4.1.1 Modeling

Most of the modeling process have been explain in chapter 3. The main difference between the generic problem and the BPP is that the cost of a column is always 1. In addition to simplify the objective of the slave problem (see example page 10), we can also derive a tighter lower bound. Recall the bound $z^\star \geq z + r^\star \kappa$, with $\kappa \geq \sum_i x_i^\star$. In the case of the BPP, the best we can choose is $\kappa = z^\star = \sum_i x_i^\star$, and the lower bound on $z^\star$ is:

$$\begin{aligned} z^\star &\geq z + r^\star \kappa \\ z^\star &\geq z + r^\star z^\star \\ z^\star(1 - r^\star) &\geq z \\ z^\star &\geq \frac{z}{1 - r^\star} \end{aligned}$$

The last step is valid only if $1 - r^\star \geq 0$, which is always verified as $r^\star < 0$ unless optimally is reached, in which case we don't need the bound. Note that this bound is always positive, as opposed to the original.

4.1.2 Choice of the slave

In the case of the BPP, the slave problem consist of filling a bin with items that have the best reduced cost, i.e. a knapsack. The first version was made using our MIP solver for the slave

problem.

$$\begin{aligned} \text{maximize} \quad & \sum_j x_i \lambda_j \\ \text{subject to} \quad & \sum_i x_i w_i \leq C, \\ & x_i \in \{0,1\} \quad \forall i \in I \end{aligned} \tag{4.1}$$

Knapsack being a widely known problem, there is a variety of specific solver available. In particular, the OR-Tools library, Google open source software suite for optimization, includes a knapsack solver[1]. The only issue of using it was that OR-Tools only uses integers, issue that can easily be solved by multiplying every value by large factor (10^6 to 10^9 will match the precision of commercial MIP solvers) and dividing the objective by that same factor.

After some tests to ensure both solvers give the same results, we have also tested their speed to use the faster. On an instance of 249 items, of weight between 250 and 500 for bins of size 1000, we get the following times over 100 generated column.

	Average	Median
MIP solver	0.388 s	0.181 s
OR-Tools	0.281 s	0.031 s

The dedicated solver is quicker than our MIP solver, especially when comparing the medians to mitigate the effect of a few longer solves (because of unfortunate ordering during branching).

4.2 Capacitated vehicle routing problem

Intuitively, while a knapsack fill a bin of a BPP, in the case of a VRP, our slave problem needs to get a path. We want a minimum reduced cost, so the objective is a shortest path. The distances of the original problem will be reduced with respect to the dual variables of the master problem, so some distance will become negative. The partition constraint specify that only elementary paths are suitable, but because of some negative edges this might not be the case as infinite negative cycles can become optimal. Finally, we can add a resource constraints to reflect the capacity constraint of a CVRP. This problem is usually described in the literature as elementary shortest path problems with resource constraints (ESPPRC) but the name can vary [3, 8, 10, 11, 12].

We are not looking for an exact solution of (3.2) but for a lower bound. Thus we do not need to have the optimal solution of (3.4) but just a lower bound. This sub-problem will need to be solved many time so we will relax some constraints to accelerate the solving.

4.2.1 Shortest path approach

A first idea would be to drop the resource constraints to only keep an elementary shortest path problems (ESPP). Shortest path is a widely studied problem [2, 9, 16] but most of the usual algorithms (e.g. Dijkstra, Floyd–Warshall) don't work with negative edges. The Bellman–Ford algorithm is capable of detecting negative cost cycle, but is not able to give a solution when

[1] `https://developers.google.com/optimization/bin/knapsack`

they are present [17]. We tried to modify the algorithm to forbid multiple use of the same edge (elementary path), but then the algorithm give a feasible solution (upper bound) an not the optimal while we are looking for an optimal or a lower bound.

This "shortest path" approach is discussed in the literature [11, 14], but the algorithms described are too complex to implement in a prototype and might not be fast enough for our needs.

4.2.2 Multi sub-tour approach

As dropping the resource constraint doesn't seems to give interesting results, we tried to drop the path constraint instead, i.e. solutions might then be composed of multiple independent sub-tours.

Our first idea was to re-use the knapsack of section 4.1.2. The bin become a tour, the weights become demands, and the objective is to maximize the sum of the dual value associated to each customers. These value are penalized by the distance to the nearest neighbor, which is a lower bound on the distance to the next customer on the path in the full ESPPRC. This formulation generate an unordered list of customer to visit.

We can acheive better bound by solving a MIP, deciding whether an edge will be used or not. Let D be the depot, C the capacity of a truck, d_{ij} the distance between customer i and customer j and λ_i the dual value associated with customer i. Let x_{ij} a decision variable set to 1 if the edge ij is used in the solution, 0 otherwise, and y_i a decision variable set to 1 if the customer i is delivered, 0 otherwise.

$$\begin{aligned}
\text{minimize} \quad & \sum_{ij} x_{ij} d_{ij} - \sum_{i} y_i \lambda_i \\
\text{subject to} \quad & \sum_{hj} x_{hi} + x_{ij} = 2y_i \qquad \forall i, \\
& \sum_{i} y_i w_i \leq C, \\
& x_{ii} = 0 \qquad \forall i, \\
& y_D = 1, \\
& x_{ij}, y_i \in \{0,1\} \quad \forall ij
\end{aligned} \tag{4.2}$$

While much slower than the knapsack model, this MIP provide better bounds. Over the ten first instances of the set A of CVRPLIB[2] (see chapter 5), using the knapsack slave produces bounds on average at 20% of the optimal, and using the MIP model produces bounds on average at 62% of the optimal.

The first constraint of (4.2) ensure the solution is a tour, but does not forbid sub-tour. To solve this issue we could use a decision variable x_{ijk}, where k is the index of the edges in our tour, and add a constraint of the form $\sum_h x_{hik} = \sum_j x_{ij(k+1)}\ \forall i,k$. But we can already guess that this model will be more complicated and also longer to solve, in particular because of the cubic number of decision variables. For our purpose, (4.2) is good enough as it provide quickly a reasonably good lower bound on the real ESPPRC. Note that the bound obtain can be reinforced by adding sub-tours eliminating constraints.

[2] `http://vrp.atd-lab.inf.puc-rio.br/`

— 5 —
Results

The objective of the project is to give LocalSolver lower bounds on model that uses sets or lists variable (see chapter 2). Our interests in the results will first be the quality of such bounds, and then the cost to get them. The BPP and the CVRP being widely studied problems, many benchmark instances can be found online, on websites like BPPLIB[1] [6] and CVRPLIB[2] [18].

5.1 Bin packing problem

5.1.1 Bounds quality

The new element of our project with respect to the literature is the use of columns generated from a local search to start the column generation process. We will look at the evolution of the bound yield by this process with respect to the number of iterations done by LocalSolver. The protocol is the following, each time LocalSolver find a better solution, it sends that solution to the column generation process, which will iterate until the minimum reduce cost is non-negative. The lower bound obtained by this process is plotted alongside the optimal value and the current objective of LocalSolver. The iteration count on the x axis are the local search iterations and not the column generation iterations, it can roughy translate to the running time of LocalSolver. The bound curve does not represent the bound evolution with respect to time, but rather the best lower bound we can obtain by starting the column generation with the current best feasible solution, i.e. if we stop LocalSolver after 27 000 iterations (the dashed line), it give a feasible solution with an objective value of 170 and the best bound we can obtain from that is 145.

Typical result will look like fig. 5.1. We can see LocalSolver objective in blue quickly getting toward the optimal value, and then a plateau where the local search spend a lot of time to slightly improve the solution. Our bound get to $\frac{2}{3}$ of the optimal quickly, and then improve to reach 80–90% of the optimal when LocalSolver reaches it. This first result is promising as the final bound is quite good, and even if the user want to stop LocalSolver quickly, the bound is still meaningful.

While the results are good for small instances, they tend to vary more on large instances. For example, fig. 5.2a show one of the worst result of the 1000 items instances, the bound only get to 54% of the optimal. But on fig. 5.2b the bound get to 94% of the optimal. The average over

[1] `http://or.dei.unibo.it/library/bpplib/`
[2] `http://vrp.atd-lab.inf.puc-rio.br/`

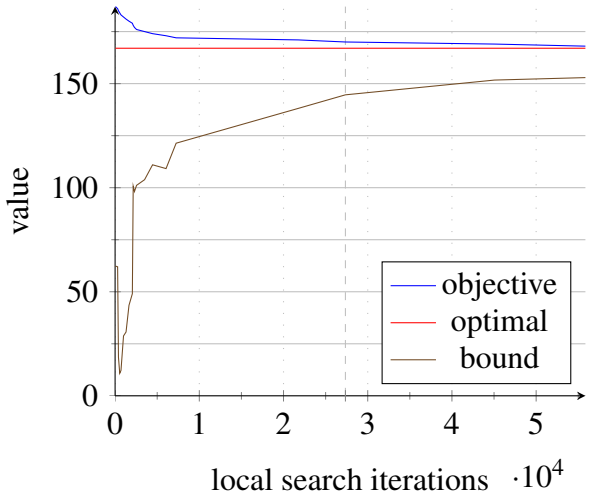
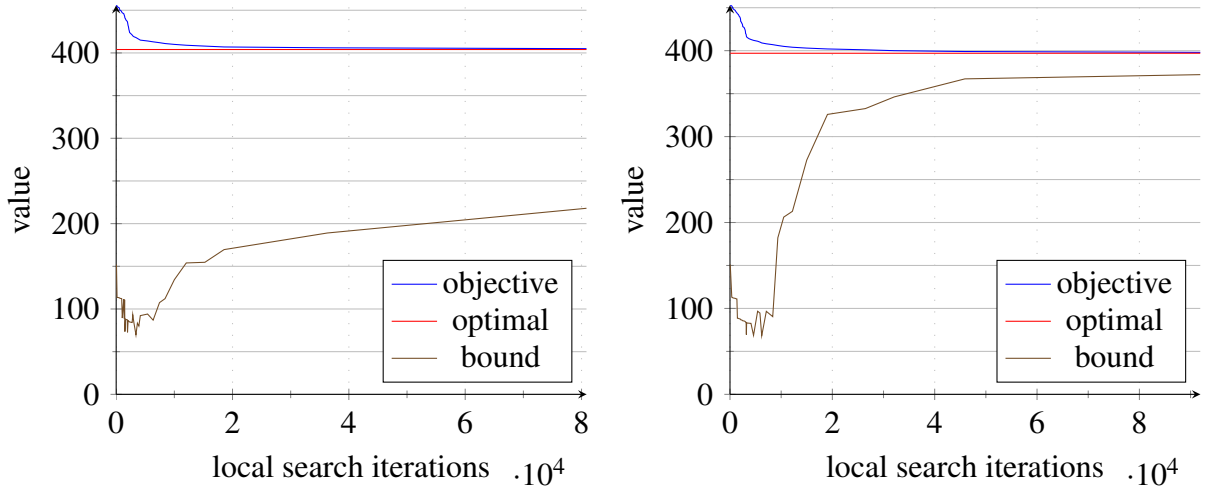


Figure 5.1 – Instance t501_01 (Falkenauer set)



(a) Instance u1000_07

(b) Instance u1000_09

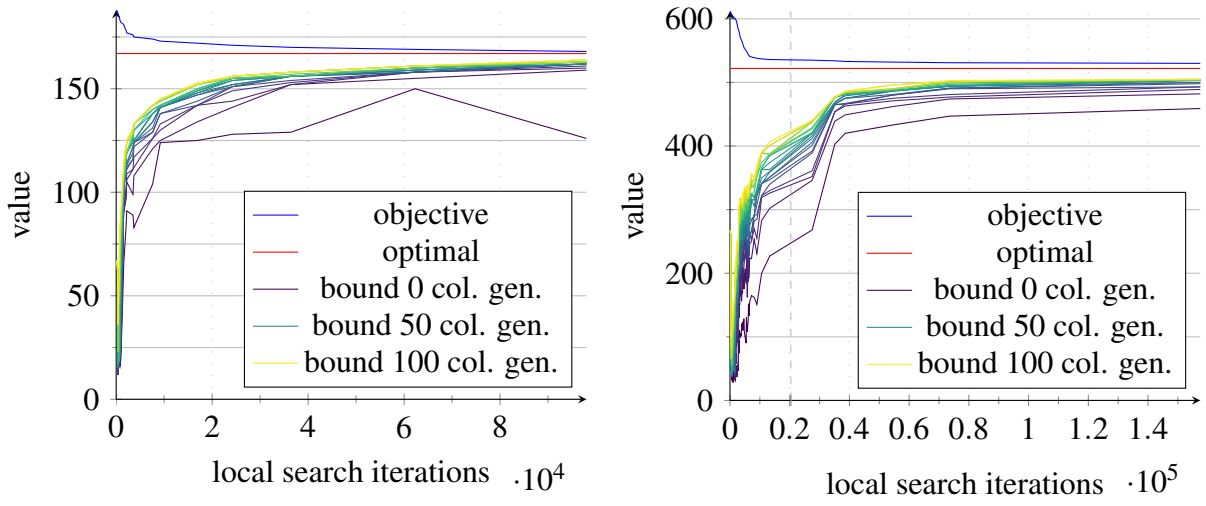
Figure 5.2 – BPP instances with 1000 items (Falkenauer set)

the 20 instance of 1000 items is a bound at 83% of the optimal. Similar results are obtained on other instances set.

5.1.2 Cost of getting the bounds

For now we have always push the column generation process to the best bound we can obtain with it. But we can also study the way the bound progresses with respect to the number of generated columns.

Figure 5.3 shows the evolution of the bounds as the number of columns generated increase from 0 to 100 (with steps of 5). We can see on fig. 5.3a that most of the progress append with the first generated columns. Even if we want to get lower bounds really quickly and don't have the time run the column generation process to its limit, generating a few columns (5–20) is already a good improvement compare to just computing the bounds. Figure 5.3b show similar results, but also highlight the importance of good firsts columns. For example, we can see that there is a



(a) Instance t501_00 (Falkenauer set)

(b) Instance csAA125_1 (Gschwind & Irnich set)

Figure 5.3 – Evolution of the bounds with respect to columns generation

much greater difference between 0 and 100 columns generated at the beginning of the graph than at the end, when LocalSolver have found better solution.

5.1.3 Comparison with a relaxed MIP

While we might be satisfied by the result of our prototype, we need to compare it with what is already available to ensure it is useful. A way to get a lower bound on a BPP is to solve its linear relaxation with a LP solver. In particular, we expect our MIP solver to give good result for small instances, but we want to compare how the two methods scale.

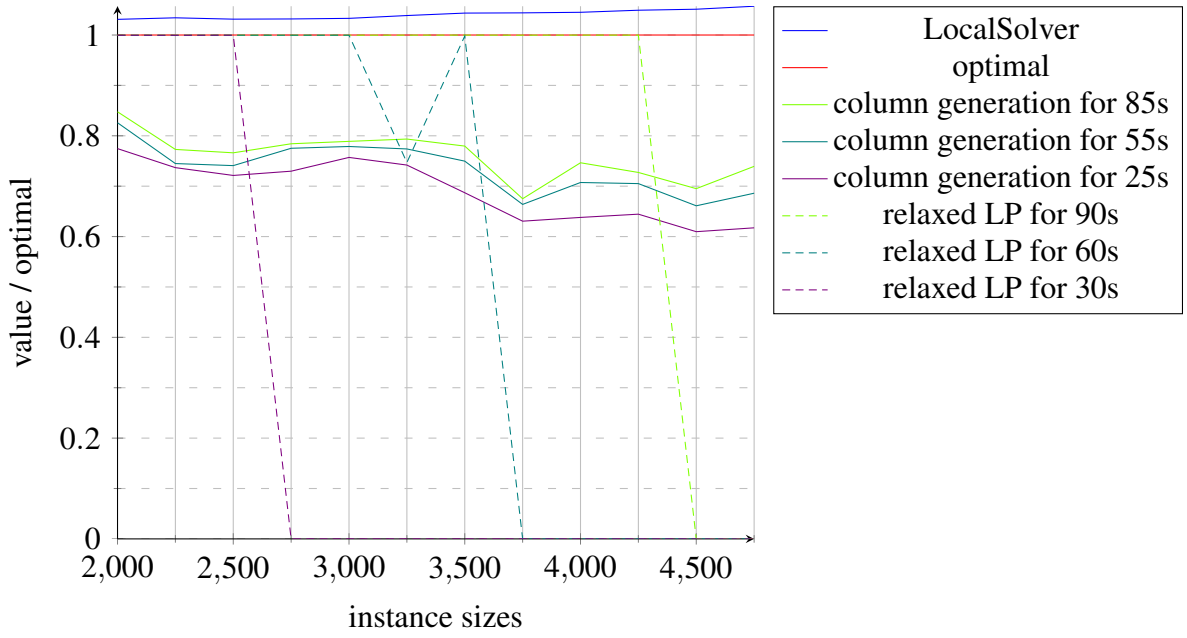


Figure 5.4 – Comparison between column generation and relaxed LP

Figure 5.4 show a comparison between the column generation prototype and a relaxed LP given to our MIP solver. Each methods was run on problems of sizes between 2000 and 5000 items, 10 instances of each sizes, randomly generated. The MIP solver was run for 30, 60 and 90 seconds. LocalSolver was run for 5 seconds to generate the first columns, and the 25, 55, and 85 second of column generation. The bounds are expressed as a ratio with respect to the optimal of the linear relaxation, as to hide differences between instance sizes.

We can observe that the instance sizes greatly affect the MIP solver as after some size it can't finish and become useless whereas the column generation bounds slowly decrease but are still interesting (60–75% of the optimal), even with big instances.

5.2 Capacitated vehicle routing problem

The main difference between the BPP and the CVRP is the formula used to compute the bound. The formula described in section 4.1.1 is a really good transformation of (3.5). Unfortunately, we can't used it for the CVRP and the best estimation of κ we can get is the number of columns in the current solution. Some nice property of the formulas for the BPP, e.g. the bounds can't be negative, are not true for the CVRP. The main consequence is that the bounds obtained at the beginning of the column generation process tend to be around -200% to -350% of the optimal (instead of a reasonable 20–30% for the BPP). This means that the bounds are useless at the beginning of the process as they are negative and 0 is then a better lower bound of a CVRP.

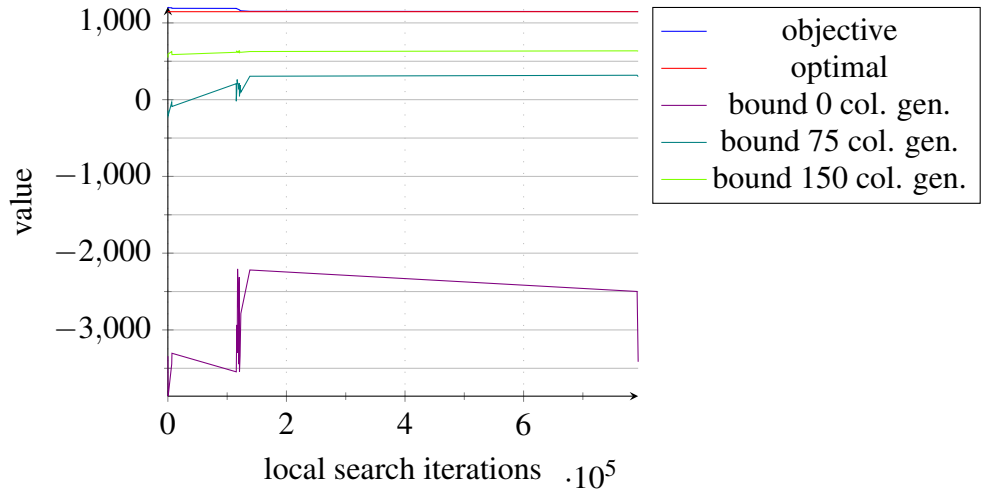


Figure 5.5 – Instance A-n45-k7 (Augerat set A)

Another difference with the BPP is a decrease of the quality of the bounds, at around 60% of the optimal for the CVRP against around 80% for the BPP. This is because the slave problem of the CVRP, the ESPPRC, is more difficult to solve than the slave problem of the BPP, the knapsack, and while we solve the exact knapsacks, we are only solving a relaxation of the ESPPRC.

Overall, we have less studied the results of the CVRP, as the objective of this project was to develop a functional prototype of column generation, and the proof of concept was already made with the results obtain on the BPP.

— 6 —
Conclusion

The research project conducted during this internship was the development of a prototype that compute bounds on large problems. Column generation is a method to solve large MIP and is particularly suited to solve BPP and VRP, the two problems studied. An advantage of this methods is that a bound can be computed at each step, thus the process might be stopped early and still provide and usable results.

The main difficulties encountered during the implementation of this project was to choose a good way to solve the slave problems. These problems need to be repeatedly solve, and thus be solve efficiently. While this is simple for the knapsack (the slave problem of the BPP), as already made solvers can be found easily, this become a challenge for the ESPPRC (the slave problem of the CVRP). The solution used for the later was to relax some constraints, making the problem easier to model and to solve, but at the cost of degrading the bounds obtained.

The results obtained with this prototype were conform to our expectations, with bounds not being perfect, at about 80–90% of the optimal in the case of the BPP (60% for the CVRP), but being fast to compute. We also confirmed the scalability of the approach, by obtaining reasonable bounds on large instance in less than 30 seconds, where our MIP solver took more than triple the time.

Overall, the prototype confirms our expectations, and bounds computation by column generations will be added to LocalSolver in the future. But a lot of work on the formulation and solving of slaves problems will be required to be more generic and tackle problems with other kind of constraints.

Bibliography

[1] M. S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear programming and network flows*. John Wiley & Sons, Hoboken, N.J, 4th ed edition, 2010. OCLC: ocn419865390.

[2] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, April 1958.

[3] Natashia Boland, John Dethridge, and Irina Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68, January 2006.

[4] Vašek Chvátal. *Linear programming*. A Series of books in the mathematical sciences. W.H. Freeman, New York, 1983.

[5] George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, February 1960.

[6] Maxence Delorme, Manuel Iori, and Silvano Martello. BPPLIB: a library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235–250, March 2018.

[7] Jacques Desrosiers and Marco E. Lübbecke. A Primer in Column Generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 1–32. Springer US, Boston, MA, 2005.

[8] Luigi Di Puglia Pugliese and Francesca Guerriero. On the shortest path problem with negative cost cycles. *Computational Optimization and Applications*, 63(2):559–583, March 2016.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[10] Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, 8(4):407–424, December 2010.

[11] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, October 2004.

[12] Stefan Irnich and Guy Desaulniers. Shortest Path Problems with Resource Constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 33–65. Springer-Verlag, New York, 2005.

[13] Marco E. Lübbecke and Jacques Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, December 2005.

[14] Axel Parmentier. *Algorithms for shortest path and airline problems*. PhD thesis, Université Paris-Est, November 2016.

[15] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A Generic Exact Solver for Vehicle Routing and Related Problems. In Andrea Lodi and Viswanath Nagarajan, editors, *Integer Programming and Combinatorial Optimization*, volume 11480, pages 354–369. Springer International Publishing, Cham, 2019.

[16] Bernard Roy. Transitivité et connexité. *Comptes rendus de l'Académie des Sciences*, 249:216–218, July 1959.

[17] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Number 24 in Algorithms and combinatorics. Springer, Berlin ; New York, 2003.

[18] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research*, 257(3):845–858, March 2017.