5
A Novel Decomposition Algorithm

5.1 Summary

The structure of the Petroleum Supply Planning problem calls naturally for some kind of decomposition. In fact, we can face this problem as two subproblems: a platform offload planning and a refinery supply problems. However, as described in Chapter 3, both of these subproblems have discrete variables in their models which excludes the application of the traditional Benders as well as the Dantzig and Wolfe decomposition as a solution method. Based on general duality ([Tin81, Wol81, Fli93]), logic theory [Hoo03, Hoo07, Faz09] and disjunctive programming [Chu04], several authors have extended the Benders decomposition method to apply to the case where the subproblems are integer programs. On the Dantzig and Wolfe decomposition side, much work has been done on using it in a branch-and-cut-and-price scheme with very positive results on some classes of problems [Bar96, Uch08, Fuk06, Rop09]. We have not tried any of this extension to our problem to find that some of these ideas cannot yet be used to solve real large scale problems or because we did not recognize a structure that could fit into the framework of these methods. Regarding the Lagrangean decomposition, we applied it to our problem without any success. The convergence of the Lagrangean multipliers has shown to be a hard task for this problem, even after testing other more sophisticated subgradient types of algorithms. Motivated by these facts, we propose a novel decomposition algorithm that borrows some ideas from the methods presented in Chapter 4 and at the same time manages to circumvent their drawback to solve our problem. It turns out that the idea of this new algorithm is not limited to the structure of our specific problem, but can also be applied to a wide range of problems. In this chapter we present some computational results on the generalized assignment and the parallel machine scheduling problems and in Chapter 7 we show how we use this algorithm to solve the Petroleum Supply Planning problem.

5.2 The Basic Idea

For ease of exposition, we consider only pure Integer Programming problems. However, everything developed in this chapter applies in a straightforward way to more general settings, as Mixed Integer programming problems. Let an Integer Program (IP) be defined as

$$
\begin{array}{lll}
 & \min & f^T x \\
(IP) & s.t. & Ax \geq b \qquad (5.1) \\
 & & Cx \geq d \qquad (5.2) \\
 & & x \in \{0,1\}^n
\end{array}
$$

where, as usual, f^T, x, b and d are vectors, A and B are matrices, all of conformable dimensions.
Let us suppose (IP) is easily solved if one of the set of constraints (5.1) or (5.2) is removed. In the spirit of the Lagrangean decomposition, we can rewrite (IP) as follows

$$
\begin{array}{lll}
 & \min & f^T x \\
(IP') & s.t. & Ax \geq b \qquad (5.3) \\
 & & Cy \geq d \qquad (5.4) \\
 & & x = y \qquad (5.5) \\
 & & x \in \{0,1\}^n, y \in \{0,1\}^n
\end{array}
$$

Which can be decomposed into

$$
\underbrace{(IP_1) \quad \begin{array}{ll} \min & f^T x \\ s.t. & Ax \geq b \\ & x \in \{0,1\}^n \end{array}}_{\begin{array}{c}\textit{Master problem} \\ \textit{Optimization problem}\end{array}}
\quad \text{and} \quad
\underbrace{(IP_2) \quad \begin{array}{ll} \min & 0y \\ s.t. & Cy \geq d \\ & y = \bar{x} \\ & y \in \{0,1\}^n \end{array}}_{\begin{array}{c}\textit{Subproblem} \\ \textit{Feasibility problem}\end{array}}
$$

In this way, an algorithm to solve (IP) would be to optimize (IP_1) and check whether or not this solution is feasible to (IP_2). In case this solution is feasible, we have found the optimum solution to (IP), otherwise we need to cut off at

least this infeasible solution and keep on doing these basic steps. We should notice that after introducing these cuts the easy structure of (IP) can be broken and become harder to solve, however this is rarely a problem as we will show in the Computational Experiments Section. As we can see, this algorithm is pretty general since no assumption was made in its description. Moreover, as the subproblem is a feasibility problem, it opens up a range of possibility to apply hybrid methods, such as combination of mixed integer programming with constraint programming. In this work we stick to the framework of mathematical programming and we show a very effective scheme to generate cuts.

(a) Cut Generation

The goal of the cut generation procedure is to cut off at least the solution that is proved not to belong to the feasible region of the problem. In the proposed decomposition algorithm, the trial solution turns out to be always integer. Thus, an easy scheme to cut off this solution is just to logically negate it, as shown in the sequel.

Let x be an optimum solution to the Master problem (IP_1), but infeasible to the subproblem (IP_2). Defining $S = \{i \in N | x_i = 1\}$, in order to cut off this solution we can build the following logical constraint:

$$\neg \left(\wedge_{i \in S}(x_i) \wedge_{i \in N \setminus S}(\neg x_i) \right)$$

Using De Morgan's Law, we can rewrite it as

$$\vee_{i \in S}(\neg x_i) \vee_{i \in N \setminus S}(x_i) \tag{5.6}$$

Which is equivalent to

$$\sum_{i \in S}(1 - x_i) + \sum_{i \in N \setminus S} x_i \geq 1 \quad or \quad \sum_{i \in S} x_i - \sum_{i \in N \setminus S} x_i \leq |S| - 1$$

To the best of our knowledge, these cuts were first proposed by Balas and Jeroslow [Bal72], where they were called Canonical cuts. They are also known in the constraint programming community under the name of No-good constraints [Hoo00]. It is well known in the literature that these cuts are very weak in practice and this is not surprising since they do not use any information about the feasible region of the problem they are cutting the solution from. In this work, we explore a better way to use the information provided by the subproblem, trying to explain why a given solution to the master problem is not

feasible to the subproblem. For this purpose, we add the logical expression (5.6) to the subproblem formulation (IP_2), as shown by (5.7), and through a separation scheme we look for the best cut possible.

$$\vee_{i\in S}\begin{pmatrix} Cy \geq d \\ y_i = 0 \\ y \in \{0,1\}^n \end{pmatrix} \vee_{i\in N\setminus S}\begin{pmatrix} Cy \geq d \\ y_i = 1 \\ y \in \{0,1\}^n \end{pmatrix} \tag{5.7}$$

The resulting cut generation problem is a formulation of a disjunctive program. This is the key element of our proposed separation procedure as described in the following.

Definition 5.1 (Separation Problem) *Given $x \in \mathbb{R}^n$, the problem of separating x from P is that of deciding whether $x \in P$ and if not, determining $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ such that $\alpha y \geq \beta,\ \forall y \in P$ but $\alpha x < \beta$*

We denote by $SEP(x, P)$ the procedure that separates an arbitrary x from a polyhedron P, returning either a set of valid inequalities for P or an empty set. In practice, we solve the optimization version of $SEP(x, P)$ that consists in finding the most violated inequality or the deepest cut.

Separation problem

Let a Disjunctive program be defined as

$$\vee_{k\in K}\begin{pmatrix} A^k x \geq b \end{pmatrix} \tag{5.8}$$

where vectors $x \in \mathbb{R}^n_+$, $b \in \mathbb{R}^n$ and matrices $A^k \in \mathbb{R}^{m\times n}$. Note that here the matrices A^k also include the bounds on variables x.

The question we want to answer is how to find valid inequalities for (5.8). But this question can be posed in a slightly different way, i.e., how to find inequalities $\alpha x \geq \beta$ dominated by (5.8). According to a classical result on linear inequalities (see Theorem 22.3 of [Roc70]) this is equivalent to

$$\vee_{k\in K}\begin{pmatrix} \alpha - \theta^k A^k \geq 0 \\ \beta - \theta^k b \leq 0 \\ \alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, \theta^k \in \mathbb{R}^m_+ \end{pmatrix} \tag{5.9}$$

Besides, as we want to find the best possible cut separating a given solution $\bar{x}$, one common objective function to this end is to look for the most violated cut $\alpha x \geq \beta$, i.e., the objective function that maximizes the difference $\beta - \alpha\bar{x}$, as shown in (5.10).

$$SEP(\bar{x}, P): \quad \vee_{k \in K} \begin{pmatrix} \max \beta - \alpha \bar{x} \\ \alpha - \theta^k A^k \geq 0 \\ \beta - \theta^k b \leq 0 \\ \alpha \in \mathbb{R}^n, \beta \in \mathbb{R}, \theta^k \in \mathbb{R}^m_+ \end{pmatrix} \tag{5.10}$$

Unfortunately, as one can observe, this separation problem is a polyhedral cone. Therefore, its objective function is unbounded. To circumvent this issue we need to truncate $SEP(\bar{x}, P)$ to obtain some meaningful results. This is performed in practice by some normalization inequalities as discussed in the sequence.

Normalization

Two important factors that influence the solution of the cut generation procedure is the objective function and the normalization used in the separation problem. In his seminal work on Disjunctive programming, Balas [Bal98] has shown that an inequality $\alpha x \geq \beta$ defines a facet of a disjunctive program if and only if the pair (α, β) is an extreme ray of the separation problem $SEP(\bar{x}, P)$. This implies that, if we want to obtain facets of a disjunctive program using the separation problem stated earlier, the normalization has to be such that the solution of the truncated problem still lies on an extreme ray of the original separation problem. Nevertheless, this is obtained only if we intersect the separation problem with a unique hyperplane, otherwise we can get a solution that is not associated with facets, as depicted by Figure 5.1.

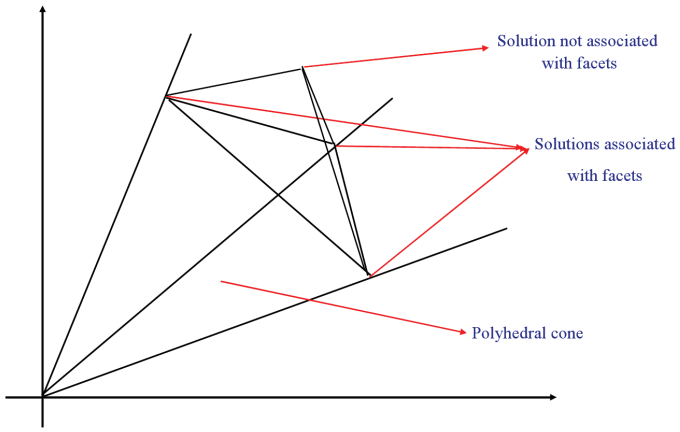

Figure 5.1: Possible outcome of different normalizations

Normalization is still a topic that should deserve some attention in the mathematical programming community. Up to now we yet do not have a normalization that guarantees, in general, to truncate the cone in (5.10) with a single hyperplane and bounds its solution. The most used normalizations found in the literature are as follows:

(i) $\beta \in \{0, 1\}$

(ii) $\sum_i |\alpha_i| \leq 1$

(iii) $\sum_k \sum_i \theta_i^k = 1$

Surprisingly enough, the only normalization that guarantees intersecting the separation problem with a unique hyperplane is (i), however, it comes with no assurance that the truncated separation problem will have a bounded solution. Balas [Bal98] has shown that if this normalization is used, the truncated separation problem has a finite maximum if and only if $\lambda\bar{x}$ is a point of the feasible region of (5.8) for some $\lambda \in \mathbb{R}_+$. This condition is satisfied by some classes of problems, such as set covering ($\beta = 1$) and set packing ($\beta = -1$), or more generally, for any combinatorial optimization problem that can be solved by replacing the polyhedron of feasible points either by its dominant or by its submissive. Regarding normalizations (ii) and (iii), they do not guarantee intersection with the separation problem by a unique hyperplane. It can be shown that the normalization (ii) corresponds to an n-dimensional octahedron, and therefore the solution of the truncated separation problem will not necessarily lie on an extreme ray of the original separation problem. In the same way, the normalization (iii) do not intersect with the separation problem by a single hyperplane either. Although its expression corresponds to a hyperplane, it does not define a hyperplane in the (α, β) space. In this work we use normalization (iii) because it performs better in practice as reported by Balas et al. [Bal96].

Upshot of the algorithm

After explaining the main points of our proposed algorithm we can make the following comments:

- If the original problem is a polytope, the proposed algorithm terminates in a finite number of steps. This is a consequence of the cut generation procedure that guarantees to cut at least the integer solutions previously found and the fact that a polytope has only a finite number of integer points.

- The optimal solution of the master problem provides a lower bound on the original problem objective function. This is clear since the master problem is a relaxation of the original problem.
- The optimal solution of the master problem in each step of the algorithm is nondecreasing for a minimization problem. In fact, at each step at least the previous solution is cut off from the master problem region. In case this solution is not unique, the solution value of the master problem in the next step may be the same, otherwise we can be sure that its optimum value would be greater than the solution value found in the previous step of the algorithm.
- The proposed cut is at least as strong as the canonical cut and frequently it is much better than it, as depicted by Figure 5.2. This observation comes with no surprise since it can be shown that the canonical cut corresponds exactly to our cut generation procedure applied to the unit hypercube, which is equivalent to completely ignore the information provided by the subproblem.

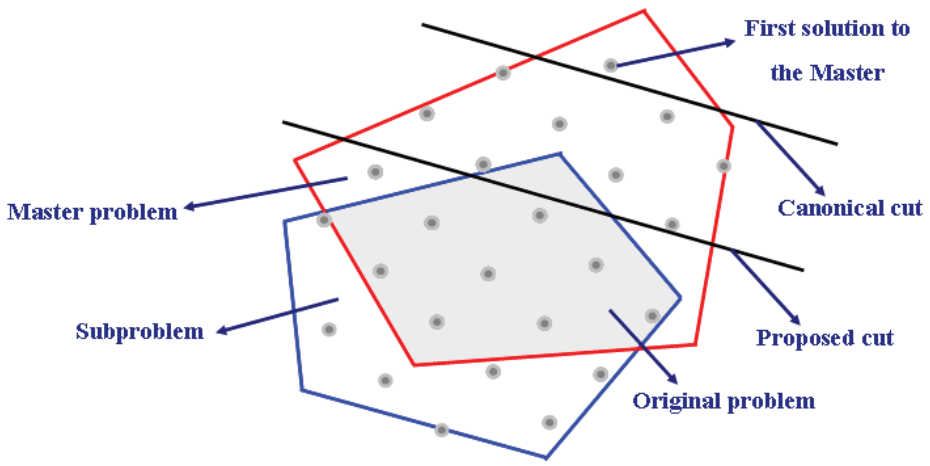

Figure 5.2: Comparison between the proposed cut and the canonical cut

5.3 Improved Idea

In real problems, one is not necessarily interested in finding the true optimal solution to a model. On the contrary, in practice one is looking for a good solution to a model guaranteed to be close enough to its true optimal solution. As mentioned earlier, the proposed algorithm provides only a lower bound on the optimal solution to a problem, and therefore the absence of upper bound information prevents the implementation of a scheme to stop our algorithm prematurely in case these bounds are close. Moreover, one

common problem of cutting plane based algorithms is the lack of progress on the lower bound value after a number of cut generation steps. In our algorithm, we observed that after a number of steps, the lower bounds obtained are often close to the optimal solution, but it suffers from the curse of multiple solutions to the master problem, as shown in Figure 5.3. As we can see in Figure 5.3, points 1, 2 and 3 in the master problem feasible region have the same objective function value, however, only point 2 is feasible and hence optimal to the original problem. From the cut shown in Figure 5.3, we can conclude that the lower bound found has exactly the same value as the optimal solution, but it can happen that we will need to cut off points 1 and 3 before reaching the optimal solution (point 2). To overcome this issue, we propose an extension of the basic algorithm using an idea from Fischetti and Lodi to repairing Mixed Integer Program (MIP) infeasibility through local branching [Fis08]. The improved algorithm consists of generating cuts until the lower bounds obtained do not have any significant improvement, and at this moment switching to the repairing MIP infeasibility method using as input the feasible solution to the master problem, which is infeasible to the subproblem and consequently to the complete problem. We claim that, in general, at this point the optimal or a very good solution is not far from the optimum solution to the master problem, as presented in Figure 5.3. In chapter VII, we show that this is the case for the petroleum supply planning problem.

In the following sections we present the Local Branching and the Repairing MIP Infeasibility ideas, and we give an outline of the whole method presented by Algorithm 3.

(a) Local Branching

The Local Branching procedure was first proposed by Fischetti and Lodi [Fis03] with the aim of improving the efficiency of an exact algorithm like Branch-and-Bound. It consists of reducing the solution search space by introducing to the model some inequalities called "Local Branching cuts", i.e., given a new incumbent solution $\bar{x}$ and a nonnegative integer parameter k (the neighborhood size), the space solution is reduced by means of the following inequalities:

$$\Delta(x, \bar{x}) = \sum_{j \in S} (1 - \bar{x}_j) + \sum_{j \in B \setminus S} \bar{x}_j \leq k \tag{5.11}$$

where, B is the set of binary variables and S is the index set of binary variables that take value of one in the solution $\bar{x}$.

The basic idea of Local Branching is to optimize over a small neighbo-

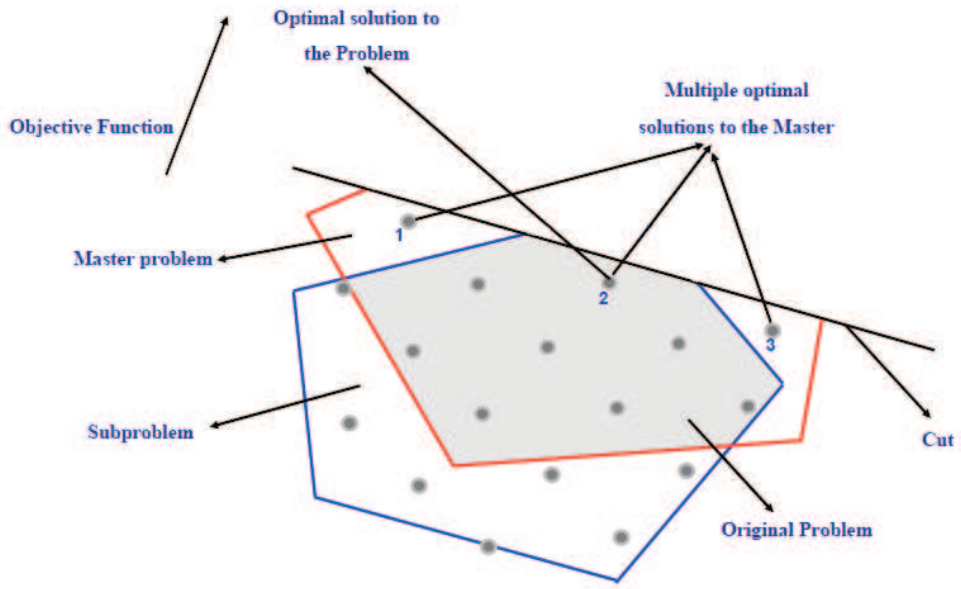

Figure 5.3: Possible limitation of the proposed algorithm

rhood, given by inequality (5.11), in an attempt to finding better solutions faster. It should be noticed that the Local Branching cuts embody the idea of a neighborhood search, similar to the idea present in most of metaheuristics, as they express that the solution at hand can be changed by at most k elements. One parameter of this procedure that needs to be determined is the neighborhood size k. There is a trade-off in choosing the value for k, as we want it to be as small as possible to make the problem easy to solve, but large enough to contain solutions better than the given one. Fischetti and Lodi suggested that a value of k in the interval [10, 20] works best in the majority of cases. Besides, in the implementation of the procedure, this parameter can be dynamically modified as the algorithm proceeds. Nowadays, the Local Branching procedure is implemented by almost all commercial solvers, and in particular, it is provided by CPLEX disguised by the name of polishing algorithm. The question one could raise is why bother implementing this algorithm. This is justified by the fact that many solvers do not provide sufficient control over their algorithms and they usually apply Local Branching to all integer variables of a given problem. Nonetheless, the experience has shown that it is sometimes advantageous to use this algorithm only with a subset of integer variables, leaving more room for the solver to find better solutions. In this work we implement a version of the Local Branching, called asymmetric, that counts only the number of variables changing from 1 to 0, as given by the expression (5.12).

$$\sum_{j \in S} (1 - \bar{x}_j) \leq k' \tag{5.12}$$

where $k' \approx \lfloor k/\theta \rfloor$ and θ is a parameter problem dependent.

We can better understand how this procedure works by its basic tree representation as given in Figure 5.4. The upshot of this method is that we build a search tree where only the nodes appended by triangles with an S (for Solver) are explored by an off-the-shelf MIP solver. We start out with an initial feasible solution $\bar{x}_1$, and we add to the model the Local Branching constraint $\Delta(x, \bar{x}_1) \leq k$, that corresponds to the node 2 in the tree. We solve the problem represented by node 2, and find the optimum solution $\bar{x}_2$ which is better than $\bar{x}_1$. At this point, we delete from our model the constraint $\Delta(x, \bar{x}_1) \leq k$, adding simultaneously the constraints $\Delta(x, \bar{x}_1) \geq k + 1$ and $\Delta(x, \bar{x}_2) \leq k$ (these steps lead us to node 4). We keep on doing these basic steps until we reach a stopping criteria. As shown in the figure, at this point we simply delete the last constraint $\Delta(x, \bar{x}_3) \leq k$ and add $\Delta(x, \bar{x}_3) \geq k + 1$, abandoning the Local Branching procedure and proceed solving the problem represented by node 7 using an MIP solver. This is the most basic description of the implementation of the Local Branching procedure. A more elaborated algorithm can be found in the paper of Fischetti and Lodi [Fis03], where they borrowed some ideas from the metaheuristic community to improve the efficiency of this implementation.

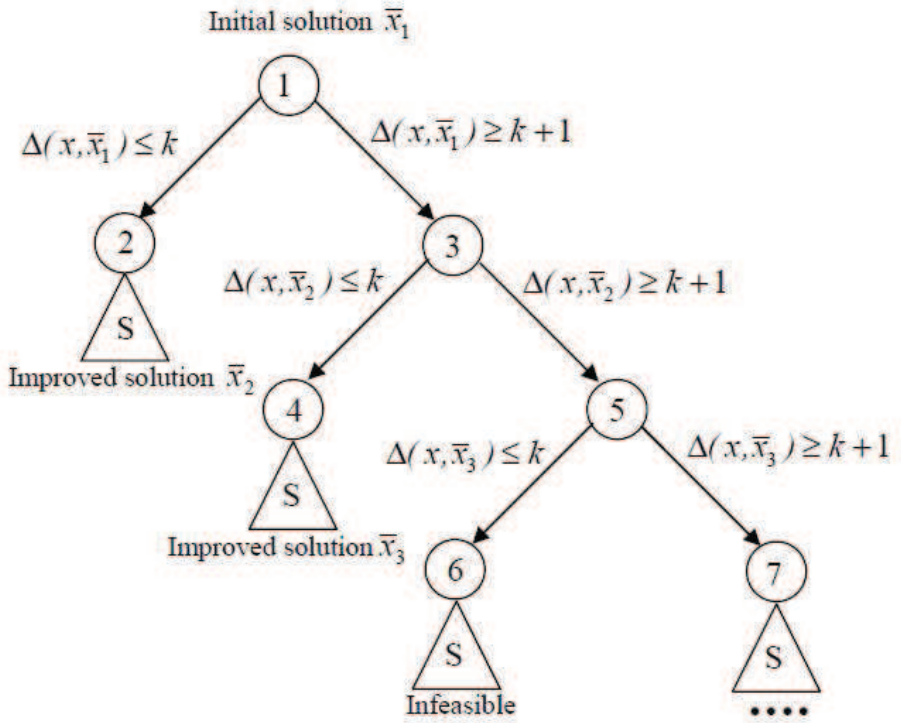

Figure 5.4: Basic Tree representation of the Local Branching procedure

In summary, the key points of this method are:

- The search is concentrated on a reduced part of the search space defined by the left nodes in the tree in Figure 5.4

- The neighborhood of a given solution is explored with an MIP solver.

(b) Repairing MIP infeasibility

This idea is a natural extension of the Local Branching procedure to deal with infeasible initial solutions. Proposed by Fischetti and Lodi [Fis08], this procedure is at the same time a very simple and powerful algorithm. We can compare it with the phase I algorithm in the Simplex scheme because it basically tries to drive an infeasible MIP solution to the feasible region by means of minimizing the sum of artificial variables corresponding to infeasibility in each constraint.
For the sake of simplicity, we consider a (IP) problem

$$\begin{array}{rll} & \min & f^T x \\ (IP) & s.t. & Ax \geq b \\ & & x_i \in \{0,1\}\ \forall i \in B \end{array} \tag{5.13}$$

Let $\bar{x}$ be an infeasible solution to (IP). Defining $T = \{i \mid a^i \bar{x} < b_i\ i \in B\}$ as the subset of constraints of (IP) that is violated by $\bar{x}$ and $\delta_i = b_i - a^i \bar{x}$ the amount of its violations. We attempt to repairing the solution $\bar{x}$ by applying the Local Branching to the following problem

$$\begin{array}{rll} & \min & \sum_{i \in T} y_i \\ (\widetilde{IP}) & s.t. & a^i x \geq b_i \qquad \forall i \in B \setminus T \qquad (5.14) \\ & & a^i x + \delta_i y_i \geq b_i \qquad \forall i \in T \qquad (5.15) \\ & & x_i \in \{0,1\}\ \forall i \in B,\ y_i \in \{0,1\}\ \forall i \in T \end{array}$$

Note that now we have readily a feasible solution to $(\widetilde{IP})$ by setting all y_i to 1. One could argue why not define y_i variables as continuous instead of binary. Fischetti and Lodi [Fis08] has shown that the Local Branching algorithm works much better with y_i binary. Similar to the phase I algorithm in Simplex, the algorithm terminates as soon as the objective function becomes zero, i.e., after all y_i are driven to zero.

Algorithm 3 A Novel Decomposition algorithm

Step 0: Set the best upper bound $bestUB = \infty$, the best lower bound $bestLB = -\infty$

while $\left|\frac{bestUB - bestLB}{bestUB}\right| > gap$ **do**

Step 1: Solve the Master problem (IP_1),

$$(IP_1) \quad \begin{array}{ll} \min & f^T x \\ s.t. & Ax \geq b \\ & x \in \{0,1\}^n \end{array}$$

if the (IP_1) infeasible **then**

Terminate.

else

Let $\bar{x}$ be the master problem solution.

Set $LB_k = f^T \bar{x}$

Check whether or not $\bar{x}$ is feasible to the subproblem (IP_2)

$$(IP_2(\bar{x})) \quad \begin{array}{ll} \min & 0y \\ s.t. & Cx \geq b \\ & y = \bar{x} \\ & x \in \{0,1\}^n \end{array}$$

if the (IP_2) is feasible **then**

Stop. Optimum solution found

else

if $bestLB \leq LB_k$ **then**

Set $BestLB \leftarrow LB_k$

end if

Solve the cut generation problem $SEP(\bar{x}, P)$ with normalization (iii)

Add row (α, β) to the Master problem $[A, b]$

end if

end if

Step 2: Obtaining feasible solutions

if $\left|\frac{bestLB - LB_k}{bestLB}\right| < \epsilon$ after η number of iterations **then**

Run Repairing MIP infeasibility with $\bar{x}$

if $\sum_{i \in T} y_i = 0$ **then**

Run Local Branching with the new $\bar{x}$

$UB_k = f^T \bar{x}$

if $bestUB \geq UB_k$ **then**

Set $BestUB \leftarrow UB_k$

end if

end if

end if

end while

5.4 Computational Experiments

In this section we present some results for the Parallel Machine Scheduling and the Generalized Assignment Problems. We implemented these models on C++ using ILOG Concert Technology compiled on Visual Studio 2008. All tests were run on CPLEX 11.2 using an Intel Centrino 2.8 GHz 4GB of RAM. The objective of these tests were to compare the efficiency in terms of computational time of the proposed algorithm in solving these problems to optimality in a 2 hours time limit, and therefore only the basic algorithm was applied.

(a) Generalized Assignment problem

The Generalized Assignment is an important theoretical problem in the Mathematical Programming area because it appears as subproblem in a large number of real applications. It concerns the maximum satisfaction assignment of jobs to machines, such that each job is assigned to exactly one machine with limited processing capacity. This problem is known to be NP-hard and although it has been around for more than four decades, it still recently attracts interest in the literature [Nau03, Yag04]. The formulation of this problem goes as follows.

Sets

- $I = 1, ..., m$: set of jobs
- $J = 1, ..., n$: set of machines

Indices

- $i \in I$: Jobs
- $j \in J$: Machines

Data

- Satisfaction level of assigning job i to machine j: $c_{i,j}$
- Capacity of machine j: b_j
- Resource consumed by assigning job i to machine j: $a_{i,j}$

Decision Variables

- $x_{i,j}$: 1 if job i is assigned to machine j; 0 otherwise.

$$
\begin{aligned}
& \max \quad \sum_{i \in I} \sum_{j \in J} c_{i,j} x_{i,j} \\
(IP) \quad & s.t. \quad \sum_{j \in J} x_{i,j} = 1 \quad \forall i \in I && (5.16) \\
& \sum_{i \in I} a_{i,j} x_{i,j} \leq b_j \quad \forall j \in J && (5.17) \\
& x_{i,j} \in \{0,1\} \ \forall i \in I, \ \forall j \in J
\end{aligned}
$$

where (5.16) are assignment constraints of jobs to machines and (5.17) are the capacity constraints on the machines.

To test the efficiency of our decomposition algorithm in solving the Generalized Assignment, we have considered the following alternative solutions:

- **CPLEX**: Model solved using CPLEX with default parameters
- **Decomp.**: Model solved by the decomposition algorithm, where the master problem is the Assignment subproblem (inequality (5.16)) and the subproblems are the $|J|$ knapsack subproblems (capacity inequality (5.17))
- **MRoc_B&P**: A Branch-and-Price algorithm implemented by Medeiros Rocha [Med09], specially designed to solve the GAP.

The instances we use in our computational tests are from Yagiura Website [Yag10], however, we only consider a subset of the groups C, D and E that at least one of the alternative solutions can find its optimum in the 2 hours time limit. The nomenclature of the problem data is as follows: The first letter corresponds to the instance group, the two first digits are the number of machines and finally the last three digits are the number of jobs.

In Table 5.1 we present a comparison of the alternative solutions to solve the Generalized Assignment problem. From these results, we cannot declare a clear winner since each method is the fastest approximately the same number of times. Nevertheless, we can draw the following conclusions:

- The algorithm **MRoc_B&P** is the fastest for the easiest instances, while the algorithm **Decomp** is the fastest for the hardest ones.
- The algorithm **MRoc_B&P** could not solve to optimality 6 out of 18 instances. The same happens with **CPLEX** in 2 out of 18 instances, while algorithm **Decomp** solves all 18 instances in the given time limit.

Table 5.1: Computational results for the Generalized Assignment

Problem Data	Objective Function	CPU(s)		
		CPLEX	Decomp	MRoc_B&P
c05100	1931	0.95	1.51	0.56
c05200	3456	2.71	8.30	1.89
c10100	1402	2.32	5.04	2.01
c10200	2806	31.64	98.14	68.20
c20100	1243	2.40	7.05	1.77
c20200	2931	63.14	63.14	23.21
c10400	5597	27.05	65.01	34.39
c20400	4782	2130.75	1039.57	7200.00*
c40400	4244	207.70	603.41	129.72
c60900	9328	7200.00*	490.99	7200.00*
d05100	6353	798.60	1033.09	7200.00*
d10100	6350	7200.00*	479.72	7200.00*
e05100	12681	7.88	17.40	19.47
e05200	24930	7.35	11.20	3.09
e10100	11577	111.17	101.02	1219.70
e10200	23307	165.70	134.04	7200.00 *
e20200	22379	239.90	143.53	4619.51
e20400	44878	555.16	2018.83	7200.00 *

* Did not find the optimal solution after 7200 seconds

(b) Parallel Machine Scheduling problem

We consider a simple scheduling problem described in the paper of Jain and Grossmann [Jai01]. This problem involves finding a minimum cost scheduling to process a set of orders using a set of dissimilar parallel machines. Processing of an order can only begin after its release date and must be completed at latest by its due date. All orders can be processed on any machines. In the following, we describe the MIP model proposed by Jain and Grossmann [Jai01].

Sets

- I: set of orders
- M: set of machines

Indices

- $i, i' \in I$: Orders
- $m \in M$: Machines

Data

- Cost of processing order i on machine m: $c_{i,m}$
- Processing time of order i on machine m: $p_{i,m}$
- Release date of order i: r_i
- Due date of order i: d_i

Decision Variables

- $x_{i,m}$: 1 if order i is processed on machine m; 0 otherwise.
- $y_{i,i'}$: 1 if order i is processed before i' on a given machine; 0 otherwise.
- $ts_i \in \mathbb{R}_+$: Start time of order i on a given machine.

$$\min \quad \sum_{i\in I}\sum_{m\in M} c_{i,m}x_{i,m} \tag{5.18}$$

$$s.t. \quad ts_i \geq r_i \quad \forall i \in I \tag{5.19}$$

$$ts_i \leq d_i - \sum_{m\in M} p_{i,m}x_{i,m} \quad \forall i \in I \tag{5.20}$$

$$\sum_{m\in M} x_{i,m} = 1 \quad \forall i \in I \tag{5.21}$$

$$\sum_{i\in I} p_{i,m}x_{i,m} \leq \max_{i\in I} d_i - \min_{i\in I} r_i \quad \forall m \in M \tag{5.22}$$

$$y_{i,i'} + y_{i',i} \geq x_{i,m} + x_{i',m} - 1 \quad \forall i, i' \in I,\ i' > i,\ \forall m \in M \tag{5.23}$$

$$ts_{i'} \geq ts_i + \sum_{m\in M} (\max_{i\in I} d_i)x_{i,m} - \sum_{i\in I} \max_{m\in M} p_{i,m}(1 - y_{i,i'}) \tag{5.24}$$
$$\forall i, i' \in I,\ i' \neq i$$

$$y_{i,i'} + y_{i',i} \leq 1 \quad \forall i, i' \in I,\ i' > i \tag{5.25}$$

$$y_{i,i'} + y_{i',i} + x_{i,m} + x_{i',m} \leq 2 \tag{5.26}$$
$$\forall i, i' \in I,\ i' > i,\ m, m' \in M,\ m \neq m'$$

$$ts_i \in \mathbb{R}_+$$

$$x_{i,m} \in \{0, 1\}\ \forall i \in I,\ \forall m \in M$$

$$y_{i,i'} \in \{0, 1\}\ \forall i, i' \in I,\ i \neq i'$$

The objective function (5.18) of this problem is to minimize the processing cost of all orders. Constraints (5.19) and (5.20) ensure that all orders start processing after their release dates and are completed before their due dates. Constraint (5.21) is an assignment constraint ensuring that one order will be processed by exactly one machine. Inequality (5.22) is a valid cut that helps tighten the LP relaxation of the problem. It is based on the fact that the total processing time of all orders that are assigned to the same machine should be less than the difference of their latest due dates and earliest release dates. Constraint (5.23) is a logical inequality that makes the connection between the assignment and the sequence variables. The underlying logic behind this constraint is that if order i and i' are assigned to machine m, then either i is processed before or after i' on machine m. Constraint (5.24) is a Big-M constraint and ensures that if the sequencing variable $y_{i,i'}$ is one, then order i' is processed after order i. The logical constraints (5.25) ensures that either i is processed before i' and vice versa. Constraint (5.26) is a logical cut that

sets $y_{i,i'}$ and $y_{i',i}$ to zero whenever orders i and i' are processed in different machines.

To test the efficiency of our approach, we have implemented the following alternative solutions:

- **CPLEX**: Model solved using CPLEX with default parameters.
- **Decomp. Assign**: Model solved by the decomposition algorithm, where the master problem is the sequence subproblem (inequalities from (5.23) to (5.26)) and the subproblems are the m machine assignment subproblems (inequalities from (5.19) to (5.22).
- **Decomp. Seq.**: Model solved by the decomposition algorithm, where the master problem is the machine assignment subproblem (inequalities inequalities from (5.19) to (5.22)) and the subproblems are the m instances of sequence subproblems (inequalities from (5.23) to (5.26)), one for each time we fix the x variables to its master solution value in a given machine.
- **Harjunkoski**: Model solved by the decomposition algorithm presented in the paper by Harjunkoski and Grossmann [Har02]. They decompose the problem in the same way as in **Decomp. Seq.**, however, the cuts are generated through the canonical cut idea.

The test instances used are from [Har02], however, we changed instances (1,2), (2,2) and (3,2) to make them harder because they were so easy before that the decomposition algorithm could solve them without the addition of any cut. Table 5.2 gives an idea of the size of each instance. We do not show the whole data of these instances in this work, but the authors will be happy to provide them upon request.

Table 5.2: Characteristic of the instances

Problem	Number of Machines	Number of Orders
1,1	2	3
1,2	2	3
2,1	3	7
2,2	3	7
3,1	3	12
3,2	3	12
4,1	5	15
4,2	5	15
5,1	5	20
5,2	5	20

Table 5.3: Computational results for the Parallel Machine Scheduling

Problem Data	Objective Function	CPLEX	Decomp. Assign		Decomp. Seq.		Harjunkoski	
		CPU(s)	CPU(s)	Nb Cuts	CPU(s)	Nb Cuts	CPU(s)	Nb Cuts
1,1	26	0.02	0.13	1	0.13	1	0.08	1
1,2	25	0.09	0.359	2	0.28	2	0.20	2
2,1	60	0.22	1.70	16	2.95	16	1.27	16
2,2	52	0.19	1.16	6	1.01	3	1.41	8
3,1	104	15.79	13.56	45	9.81	18	9.10	40
3,2	91	3.33	8.38	24	1.80	2	4.02	12
4,1	116	15.49	5.31	12	-	-	3.27	14
4,2	105	2.50	2.45	6	-	-	1.36	4
5,1	159	*	16.36	69	-	-	9.64	68
5,2	144	*	20.28	35	-	-	6.95	19

* Did not find the optimal solution after 7200 seconds
- Cut off the optimal solution due to numerical instability

Table 5.3 shows the computation times for all implementations to solve the problem as well as the number of cuts generated by the decomposition algorithms. As we can see, the time spent to solve the instances shows mixed results, where **CPLEX** is faster for small instances while the **Harjunkoski** decomposition algorithm is faster for large instances. Regarding the number of cuts, our decomposition algorithm **Decomp. Seq.** generates the smallest number of cuts to solve the problem as far as the numerical instability does not become a problem. This is an important fact since the cuts generated by our algorithm are more time consuming than **Harjunkoski**, and to beat the latter in terms of computational time we need to add fewer cuts. We believe that this would happen for the large instances if it were not by the numerical instability issue of our cut generation procedure. Concerning the decomposition algorithm **Decomp. Assign**, although it did not present the numerical instability problem of the **Decomp. Seq.**, its computational time and its number of cuts generated are always worse than the best method in each criterion. This shows the importance of choosing the best way to decompose a problem. In the case of the **Decomp. Assign**, the subproblem sets apart to generate cuts is an assignment problem that has the integrality property and consequently leads to weaker cuts. This is similar to what happens with others decomposition algorithms, in which the problem kept after dualizing (Lagrangean relaxation) or the problem convexified (Dantzig and Wolfe) cannot be so easy in order to get stronger results.

An interesting comparison that can be made is related to the cuts generated by each decomposition algorithm. Table 5.4 shows an example for instance (2,2). We can see that, for this small example, the **Decomp. Seq.** and **Harjunkoski** decompositions have generated two identical cuts, however, one can also observe that cut (S_2) is stronger than (H_2), and in fact, the former can be obtained by lifting the latter. As a matter of fact, in solving instance (2,2), the cut (S_2) generated by **Decomp. Seq.** is equivalent to $(H_2, H_4, H_5$

and H_6) altogether. As for the cuts generated by **Decomp. Assign**, they are not related to the other methods and this is not surprising since they are generated by a different subproblem. Nonetheless, it also shows how general our cut generation procedure is, since there is no restriction on the signs of the coefficients as well as the type of inequalities that can be obtained.

Table 5.4: Comparison of cuts generated from different decomposition algorithm

Decomp. Assign	Decomp. Seq.	Harjunkoski
$(A_1) : x_{0,0} + x_{0,2} + x_{2,0} + x_{2,2} + x_{6,0} + x_{6,2} \geq 1$	$(S_1) : x_{0,1} + x_{2,1} + x_{6,1} \leq 2$	$(H_1) : x_{0,1} + x_{2,1} + x_{6,1} \leq 2$
$(A_2) : x_{0,0} + x_{0,1} + x_{1,0} + x_{1,1} + x_{3,0} + x_{3,1} \geq 1$	$(S_2) : x_{1,2} + x_{2,2} + x_{3,2} + x_{5,2} + x_{6,2} \leq 2$	$(H_2) : x_{1,2} + x_{3,2} + x_{4,2} + x_{5,2} \leq 3$
$(A_3) : x_{0,0} + x_{0,2} + x_{5,0} + x_{5,2} + x_{6,0} + x_{6,2} \geq 1$	$(S_3) : x_{0,2} + x_{1,2} + x_{3,2} \leq 2$	$(H_3) : x_{0,2} + x_{1,2} + x_{3,2} \leq 2$
$(A_4) : x_{0,0} + x_{0,1} + x_{3,0} + x_{3,1} + x_{4,0} + x_{4,1} \geq 1$		$(H_4) : x_{0,2} + x_{3,2} + x_{4,2} \leq 2$
$(A_5) : x_{0,0} + x_{0,1} + x_{3,0} + x_{3,1} + x_{4,0} + x_{4,1} \geq 1$		$(H_5) : x_{0,2} + x_{1,2} + x_{5,2} \leq 2$
$(A_6) : x_{0,0} + x_{0,1} + x_{1,0} + x_{1,1} + x_{4,0} + x_{4,1} \geq 1$		$(H_6) : x_{0,2} + x_{1,2} + x_{3,2} \leq 2$
		$(H_7) : x_{0,2} + x_{1,2} + x_{4,2} \leq 2$
		$(H_8) : x_{0,1} + x_{4,1} + x_{6,1} \leq 2$

Regarding the numerical instability issue of the **Decomp. Seq.** decomposition, we have tracked all the steps of the algorithm and have found out that the problem matrix of the separation problems turned out to be ill-conditioned. To cope with this problem, every time we solve the separation problem, we start by scaling its coefficient matrix in the attempt to bring down its condition number. This has allowed us to tackle instances (3,1) and (3,2), but it was not enough to cover all instances, and we believe that a more in-depth study needs to be done in order to definitely solve this problem. It should be noted that this is not an uncommon problem, as shown by the example in Figure 5.5 solved using CPLEX. The original problem has an optimum solution equal to 3, however due to the rounded data, CPLEX ends up finding an incorrect and infeasible optimum solution equal to 2.

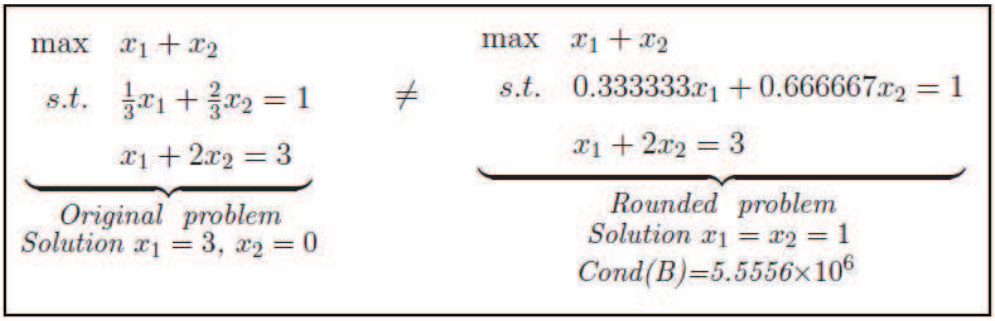

Figure 5.5: Effect of the problem matrix condition number on the optimum solution found

5.5 Conclusions

In this chapter we presented a new decomposition algorithm based on the idea of decomposing a problem through copying variables and generating cuts via disjunctive programming. Our main goal was to propose a method that could overcome the difficulties of traditional decomposition methods in dealing with pure integer programming problems. The proposed decomposition method turned out to be quite general and we proved this feature by applying it to two problems from the literature, namely, the Generalized Assignment and the Parallel Machine Scheduling problems. As mentioned, our focus was to show how the proposed algorithm could be used to solve other problems than the petroleum supply planning problem. However, the performance results also showed that this method can be a viable option to solve some hard instances of the problems tested. It is important to observe that these results were obtained by a first implementation of this decomposition algorithm and we believe that we can further improve its performance if a great care in its coding is exercised.