HYBRID METHODS FOR THE MAINTENANCE SCHEDULING OF GENERATING UNITS PROBLEM

Nuno Gomes[1], Zita Vale[1], Carlos Ramos[1], Manuel Cordeiro[2]

GECAD – Knowledge Engineering and Decision Support Group
Institute of Engineering – Polytechnic of Porto
Porto, Portugal
ngomes,zav@dee.isep.ipp.pt, csr@dei.isep.ipp.pt

Electrical Engineering Section
University of Trás-os- Montes e Alto Douro
Vila Real, Portugal
cordeiro@utad.pt

Abstract – Real world search and optimization problems are usually characterized by having large search spaces, being difficult to model and being constantly changing. Constraint Logic Programming (CLP) has proved to be a good tool to model and solve this type of problems despite some efficiency problems. It seams that the problem is related to some lack of a "global perspective" of the constraints in the search process. One way of overcoming this problem is to hybridize Constraint Logic Programming with Linear Programming (LP) in order to enhance the solving process by finding feasible solutions early in the search by "global reasoning". In this work we propose 4 hybrid methods for solving a Maintenance Scheduling of Generating Units problem. The main idea is to take the advantages of both CLP and LP avoiding their problems. Some initial tests show that for some conditions all the hybrid methods can perform better than CLP alone.

Keywords: Constraints Logic Programming, Linear Programming, Hybrid Methods, Maintenance Scheduling

I. INTRODUCTION

Real-world search and optimization problems are usually characterized by 3 important issues:

1. They are large;
2. They are complex;
3. They are constantly changing.

Due to these characteristics, modelling, solving and maintenance processes are very hard. On one side, a good tool is needed to model a large number of heterogeneous constraints, to be capable of producing a simple, understandable and flexible model. On the other hand, we need an efficient tool to correctly search a large neighbourhood. This requirements are difficult to meet, namely with only one method or technique. Some methods are very good in the modelling process and others for the search, but they cannot perform equally well on both at the same time.

In the past the trend was to continually improve each method or technique, but the recent trend is to combine methods or techniques with different and hopefully complementary characteristics, in order to achieve a better global tool. A very good and successful example of this combination is the hybridization of Constraint Logic Programming (CLP) with Linear Programming (LP) in general or Integer Programming (IP) and Mixed Integer Linear Programming (MILP).

Within CLP the problem model is built in terms of domain variables and constraints. Recent CLP systems as [1] provide a large set of expressive constraints that make the modelling process an "easy" issue. Problem models are small and very close to a natural description and therefore simple to modify, and maintain. CLP relays on branching to explore the search space, seeking for the optimal solution, and relays on Constraint Propagation to remove infeasible values on the variable domains, avoiding inconsistent solutions and speeding the search. This idea works well on medium size highly constrained problems but not so well on less constrained and large size ones. It seams that the problem is related to a lack of an "global perspective" [2] at the constraints.

On the contraire, on LP approach all the constraints are evaluated simultaneously. LP can enhance the solving process by finding feasible solutions early in the search by "global reasoning," i.e., solutions of an LP relaxation, similarly providing stronger bounds that accelerate the proof of optimality, and providing reasons for failure or a poor solution [2]. Unfortunately all the constraints in LP must be linear equalities or inequalities. This imposes a severe restriction on the expressiveness of LP as a modelling language. Modelling can be hard and models may require very large numbers of variables and constraints, becoming difficult to change and maintain.

Considering the comparison made in [2], where it is pointed that CLP can accelerate the search for a solution by:

- ✓ Reducing variable domains (and in particular by proving infeasibility);
- ✓ Tightening the linear relaxation by adding bounds and cuts in addition to classical cutting planes;

- ✓ Eliminating search of symmetric solutions, which are often more easily excluded by using symbolic constraints;

and Linear Programming (LP) can enhance the solver by:

- ✓ Finding feasible solutions early in the search by "global reasoning" i.e., solution of an LP relaxation;
- ✓ Similarly proving infeasibility earlier in the search;
- ✓ Similarly providing stronger bounds that accelerate the proof of optimality;
- ✓ Providing reasons for failure or a poor solution, so as to produce nogoods;

it seams that CLP and IP are complementary and both eligible for hybridization. We can use CLP to model the problem and tightening the linear relaxation and IP to find feasible solutions early using "Global Reasoning".

In this work we propose some CLP/IP hybrid methods in order to solve the Maintenance Scheduling of Generating Units problem (MSGU).

II. Previous Work

The integration of CLP with Linear Programming has proved to be fruitful. Several schemes of integration where tried. Apart of performance issues, they differ mainly on the class of problems they are suitable to. In the literature we can find different classifications of the integration schemes. An analysis of these classifications allows us to identify two main differentiation issues, namely modelling and decomposition. This means that integration schemes differ on the way they decompose the problem and build the model.

Some schemes use double modelling, building two similar models, one for the CLP solver and the other for the MILP solver. Then, both solvers cooperate, communicating trough some mechanism in order to solve the problem. One example can be found in [3] where "shadowed variables" are used to establish a two way communication between the CLP model and a similar MILP model. The CLP solver detects infeasibilities and relaxations that pass trough the variables to the MILP solver. In turn, the MILP solver tightens the variables bounds. An extension of this scheme is proposed in [4]; other examples can be found in [5,6].

On the same line, some schemes decompose the problem considering their structure, one suitable to the CLP solver and the other to the MIP solver. Different variations exist depending on the way the solvers cooperate to solve the master problem. For example, in [7] and [8], the authors present a scheme suitable for a class of problems, where clearly two sub-problems exist, one suitable for the CLP solver and other for the MILP solver. The MILP solver tries to find the optimal solution for one sub-problem and then the CLP solver tries to find a feasible solution for the other sub-problem. If the solution does not exist cutting planes are added to the MILP sub-problem and the process is iterated. This idea is showed using a multi-machine scheduling problem where the assignment of tasks to machines is modelled as a MILP and the sequencing of the tasks to machines is modelled as a CLP.

Finally, some schemes use a mixed approach, decomposing the problem at different levels, for example dividing the problems variables or constraints, and building relaxed problems or simply building several sub-problems. Both solvers are then used to solve the smaller and simpler problems in order to find an optimal solution to the master one. For example, in [9], [10], [11], [12] a scheme is presented relaying in the problem structure. On that scheme some constraints in the form of conditionals link the discrete and continuous elements of the problem. A model has the form:

$$\min \quad cx \quad s.t. \; h_i(y) \rightarrow A^i x \geq b^i, i \in I, y \in D, x \in R^n$$

where y is a vector of discrete variables and x is a vector of continuous variables. The antecedents $h_i(y)$ of the conditionals are constraints that can be treated with CP techniques. The consequents are linear inequality systems that can be inserted into an LP relaxation. A linear constraint set which is enforced unconditionally may be so written for convenience, with the understanding that it can always be put in the conditional form. A so called Mixed Logical/Linear Programming (MLLP) problem is solved by branching on the discrete variables. The conditionals assign roles to CP and LP: CP is applied to the discrete constraints to reduce the search and help determine when partial assignments satisfy the antecedents. At each node of the branching tree, an LP solver minimizes cx subject to the inequalities $A^i x \geq b^i$ for which $h_i(y)$ is determined to be true. This delayed posting of inequalities leads to small and lean LP problems that can be solved efficiently. A feasible solution is obtained when the truth value of every antecedent is determined, and the LP solver finds an optimal solution subject to the enforced inequalities. Computational tests reported in [13] suggest that an MLLP framework not only has modelling advantages but can often permit more rapid solution of the problem than traditional MILP solvers. Another example can be found in [14], where two integration schemes are presented. In the first the LP part of the problem is incorporated into a CLP framework as a nonprimitive constraint. Thus LP becomes a constraint propagation technique. It accesses domains in the form of bounds from the constraint store and adds new bounds obtained by minimizing and maximizing single variables. In the second scheme linear inequalities are made primitive constraints. The constraint store contains continuous inequality relaxations of the constraints but excludes integrality conditions.

Other successful integration schemes can be found in [13] and [14].

In this paper we test 4 schemes of integration that include ideas from some of the works presented above.

The main objective is to evaluate what is the best scheme for the MSGU problem.

III. THE MSGU PROBLEM

Maintenance scheduling of generating units is one of the important problems of power system operation. The generating unit outage has a great effect on system reliability, unit availability, and production costs. Past experiences showed that effective scheduling can save a considerable amount of operational costs, helping the electricity companies becoming more competitive in terms of energy price while increasing system reliability. Another important goal of MSGU is to minimize the risk of the capacity shortage throughout the year.

The MSGU problem consists of determining, for each predicted maintenance task, a specific start time in the scheduling horizon (e.g. one year), while satisfying the system constraints and maintaining system reliability. This objective should be accomplished while some criteria are optimized (e.g. the sum of operation and maintenance costs is minimized) [15].

The MSGU can be a complex problem. Usually it has associated a large and complex set of problem constraints. These restrictions can usually be summarized as follows:

- **Resource Constraints** – These constraints ensure that all the resources needed to perform the maintenance task are available during maintenance. Some of the resources are: maintenance teams, vehicles, replacement parts, etc;
- **Time Constraints** – These constraints impose, for each unit to be maintained, the valid maintenance periods. For example, in winter, some units should not be maintained due to the possible bad weather conditions;
- **Precedence Constraints** – Some units, or group of units, should be maintained before/after other units, or groups. These constraints express that;
- **Capacity Constraints** – These constraints impose that the generation capacity of each unit is never exceeded;
- **Demand Constraints** – These constraints ensure that the generated energy fulfills the demand;

A real MSGU problem has more than one solution. If we consider an objective function (cost function) *f*, then there are one or more solutions for which the value of the function is maximal (or minimal for a minimization problem). In most works, the MSGU problem objective function is a sum of cost terms that is minimized. These terms are usually the total Production Costs (PC) and total Maintenance Costs (MC) for the schedule horizon. PC represent, mainly, the fuel and operational cost, and are usually different for each unit and period (e.g. the fuel cost may vary during the year). MC associate all the maintenance costs, which are usually different for each unit and can be also different for each period (e.g. hydroelectric units are cheaper to maintain during periods of low water flow). Other objective functions can be used, see [16] for a literature review.

A. General Problem Model

The MSGU has several parameters that represent the problem data, and can be summarized as follows:

T	Available maintenance periods in the schedule horizon
U	Units to be maintained
$[e_i, l_i]$	Valid time window for the maintenance task i, being e_i the earliest start period and l_i the latest start period.
ar_t	Maximum number of units which can be maintained simultaneously in period t
m_{it}	Maintenance cost of unit i in period t
c_{it}	Production cost of unit i in period t
k_i	Maximum power generation capacity of unit i
d_t	Power demand in period t
r_i	Duration of maintenance task of unit i
I	Set of pairs of units which cannot be maintained simultaneously
Pr	Set of pairs that have precedence requirements

We also define the following variables:

S_i	Starting maintenance period of unit i
E_{it}	E_{it}=1 if unit is in maintenance in period and E_{it}=0 otherwise
P_{it}	Power generation of unit i in period t (0 if the unit is in maintenance)

A valid maintenance schedule must meet the following constraints or domain requirements, which arise naturally from the problem definition:

$$1 \le S_i - r_i \le T \qquad \forall i \in U \qquad (1)$$

$$e_i \le S_i \le l_i \qquad \forall i \in U \qquad (2)$$

$$P_{it} \le k_i \qquad \forall i \in U, \forall t \in T \qquad (3)$$

$$\sum_i P_{it} \ge d_t \qquad \forall t \in T \qquad (4)$$

$$\sum_i E_{it} \le ar_t \qquad \forall t \in T \qquad (5)$$

$$S_i + r_i \le S_j \vee S_j + r_j \le S_i \qquad (i,j) \in I \qquad (6)$$

$$S_i + r_i \le S_j \qquad (i,j) \in P_r \qquad (7)$$

$$S_i, E_{it}, P_{it} \in \mathbb{N} \qquad (8)$$

Considering constraint classification from the previous section, we can say that 1 and 2 are time constraints defining the valid maintenance period; 3 is the demand constraint and guarantees that the demand for each period is always fulfilled; 4 is the capacity constraint, and ensures that the maximum capacity of each unit is never exceeded. Finally, 5, 6 and 7 are resource constraints; 5 guarantees that the number of available resources for the period is not exceeded; 6 and 7 ensures, respectively, that two specific tasks are not scheduled simultaneously, and that one specific task is scheduled before other.

Note that E_{it} and S_{it} must be two way synchronized. This meaning that, whenever one variable is changed, the other should be updated according. This synchronization process is not an easy task. Formally we can define that if S_i changes:

$$S_i = a \Rightarrow \begin{cases} E_{it} = 1 & a \le t < a + r_i \\ E_{it} = 0 & otherwise \end{cases} \quad (9)$$

$$S_i \neq [a,c] \Rightarrow E_{it} = 0 \quad \forall t \in [b,c] \wedge a + r_i = b \wedge b \le c \quad (10)$$

If E_{it} changes:

$$E_{ib} = 0 \Rightarrow S_i \neq [a,b] \quad \forall t \in [a,b] \wedge b - r_i < a \le b \quad (11)$$

In order to complete the problem model we need to define an objective function that should be minimized or maximized. Our objective function is the following:

$$\sum_i \sum_t c_{it} \times P_{it} + \sum_i Mc_i \quad (12)$$

The objective of our search procedure is to minimize the above function: total maintenance and production costs.

IV. The Hybrid Scheme

As we said in section I, CLP is a good modeling tool. We can easily model the above constraints using predefined structures usually available in CLP systems as ECLIPSE [1]. CLP can also be a good search tool regarding the right heuristics to make variable enumeration are chosen (search strategy), the search space is not large, and the chousen constraints and respective propagation algorithms perform good domain reduction without consuming too much time. Yet these requirements are not easily fulfilled. Even then, CLP misses some capacity of making global reasoning when seeking for the optimal solution. On the other hand, LP has this capacity, although with little modeling capabilities and efficiency problems when variable domain is discrete (MILP or IP problems). In this work we explore a general idea that tries to the take advantages of both CLP and LP, avoiding their problems. The idea is represented in the scheme of Figure 1.

As shown in Figure 1, the problem is modelled using CLP. Generally all the CLP systems have a large set of available pre-defined constraints that allows to easily model the problem. Besides the primitive constraints over linear terms, there are also the named *"Global Constraints"* [17], that are more complex structures suitable for certain type of problems, as scheduling ones. One example available in ECLIPSE is:

disjunction(?Start1, +Duration1, ?Start2, +Duration2, ?Flag)

This constraint can be used to model two non-overlapping tasks with known durations. Given the starting times and durations, this constraint uses constructive disjunction to remove all inconsistent values from the domains of Start1 and Start2. This Global constraints can be used to model the constraint (6) defined in section III.A.

Another example of an useful global constraint is:

cumulative(StartTimes, Durations, Resources, ResourceLimit)

The declarative meaning is: If there are N tasks, each starting at a certain start time, having certain duration and consuming a certain (constant) amount of resource, then the sum of resource usage of all the tasks does not exceed ResourceLimit at any time. This global constraint can also be used to model the constraint (5) defined in section III.A.

We refer to [18] to a more detailed description of the MSGU CLP model.

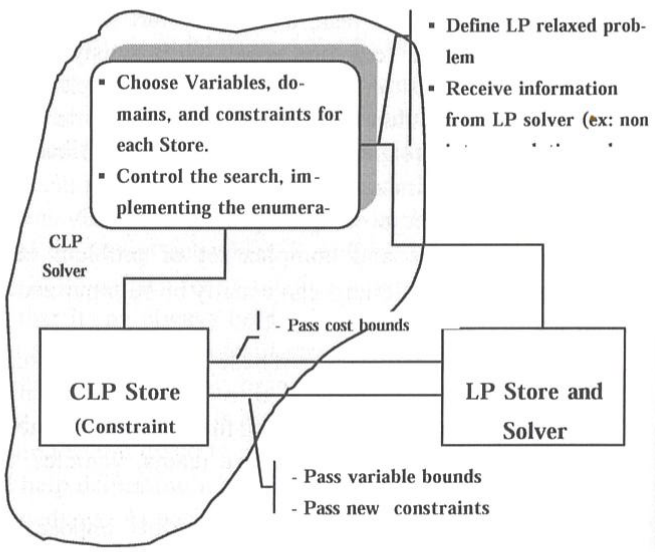

Figure 1- General Integration Scheme

After we have the CLP model, including all the problem constraints or not, some of the constraints are passed to a LP system as XPRESS-MP [19] as linear inequalities. One of the relaxed constraints is integrallity.

The search then starts by the enumeration of a particular variable on the CLP side. If necessary, after the constraints propagation, additional inequalities corresponding to new variables bounds are passed to the LP solver. The LP solver is then called to find the optimal solution of the relaxed problem. The solution cost is passed to the CLP solver and becomes the new lower bound of the non relaxed problem.

Other information can be passed from the LP side to the CLP one. This information can be used to help selecting of the next variable and respective instantiation value.

The success of this scheme depends on the trade-off between the "quality" of the LP solution and the time necessary to find it. So it is important to determine which constraints to be passed to the LP solver and when to invoke it. Unnecessary calls to the LP solver consume computation time, without producing bound improve-

ments. Nevertheless, an useful call can help the solver to earlier prune bad search tree branches.

V. SCHEME INSTANCES

We have tried 3 methods based on the previous scheme. All these methods are inspired on the works presented in [6] and [8].

A. Simple Bounds

In the Simple Bounds (SB) method only variable bounds are passed from the CLP solver to the LP solver. The CLP model contains all the problem constraints (1-11) but without including the ones related to the cost function. The method follows the following steps:

1. Convert the constraints (1-5, 10-12) to linear inequalities and pass them to the LP Solver;
2. Call the LP solver to calculate the problem cost lower bound;
3. Select a variable (the one with the smallest domain size);
4. Select a value from the variable domain (the smallest one) and use it to instantiate the variable. Propagate the constraints. If a fail occurs backtrack and choose another value for the first variable where alternative exist;
5. Convert the new variable bounds to linear inequalities and pass them to the LP solver. Call the LP solver to find the optimal solution to the relaxed problem;
6. Jump to step 3 until all the variables are instantiated;
7. If alternatives exists in previous choices impose a new constraint for the cost variable to be less than the best cost so far, and continue point 8, else end search;
8. Backtrack to the point 3 and compare the new upper bound (best cost so far) with the cost returned by the LP solver. If the new upper bound is smaller than the cost returned by the LP solver then fail, else remove the value from the variable domain and jump to step 4;

B. Informed choice

The *Informed Choice* (IC) method is similar to the previous. The difference is on the "value selection heuristic" that uses information from the LP Solver to decide which value to choose. This way, the selection heuristic selects the value based on the solution problem returned by the LP solver relaxed problem.

Within this method we have tried two variations, differing on the variables for which the choice is made. In the first variant, named IC1, branching is made over a variable that represents the total maintenance cost for each unit. Formally:

$$Tmc_i = \sum_t E_{it} * m_{it}$$

This new variable requires some new connection constraints to relate Tmc with E_{it}.

In the second variant, named IC2, branching is made over the S_i variable.

Both IC1 and IC2 method follows the following steps:

1. Convert the constraints (1-5, 10-12) to linear inequalities and pass them to the LP Solver;
2. Call the LP solver to calculate the problem cost lower boun;
3. Select a variable (with the smallest domain size, among the Tmc set for the IC1 case and among the S_i set for the VF case);
4. Try to instantiate the variable with the value equal to the solution value returned by the LP solver. Propagate the constraints. When a backtrack occurs due to a posterior fail, impose the following constraints, respecting the shown order:
   a. Var<LP solution value
   b. Var>LP solution value
   c. indomain(Var)

   Where indomain(Var) tries to instantiate Var with all the domain values;
5. Convert the new variable bounds to linear inequalities and pass them to the LP solver. Call the LP solver to find the optimal solution to the relaxed problem;
6. Jump to step 3 until all the variables are instantiated;
7. If alternatives exist on previous choices, impose a new constraint for the cost variable to be less than the best cost so far, and continue to point 8, else end search;
8. Backtrack to point 3 and compare the new upper bound (best cost so far) with the cost returned by the LP solver. If the new upper bound is smaller than the cost returned by the LP solver then fail, else remove the value from the variable domain and jump to step 4.

C. Verify Feasibility

In the Verify Feasibility (VF) method we adapt an idea of [8]. That work proposes the decomposition of the problem in two sub-problems, one suitable for a MILP solver and the other to the CLP solver. The MILP solver is called to find the optimal solution of the respective sub-problem. After, the CLP solver is called to solve a feasibility problem that extends the partial MILP solution to a global optimal solution. The proposed method it is only suitable for problems where decomposition is possible. Here we propose an adaptation of the method suitable for all kind of problems.

The idea is to build a MILP problem that considers only some of the constraints. The result is a simpler, but relaxed problem. The CLP model includes all the constraints and the CLP solver controls the search.

At each step of the search, the MILP solver tries to find an optimal solution subject to the relaxed constraints, plus the variable bounds passed from the CLP solver. Before continuing the search, CLP solver verifies if the returned solution is feasible for all the constraints. If it is, the search could stop and the solution cost is the new upper bound for the problem. The search can continue trying alternative values for the previous instantiations. The performance of the method depends on the trade-off between the number of constraints passed to the MILP solver (quality of the solution) and the time necessary to solve the corresponding relaxed problem.

The method has the following steps:

1. Convert the constraints (1-5, 10-12) to linear inequalities and pass them to the LP Solver;
2. Call the LP solver to calculate the problem cost lower bound;
3. Select a variable (that with the smallest domain size) ;
4. Select a value from the variable domain (the smallest one) and use it to instantiate the variable. Propagate the constraints. If a fail occurs backtrack and choose another value for the first variable where alternative exist;
5. Convert the new variable bounds to linear inequalities and pass them to the LP solver. Call the LP solver to find the optimal solution to the relaxed problem;
6. Verify if the solution returned by the LP solver is feasible. In affirmative case jump to point 8, else continue;
7. Jump to step 3 until all the variables are instantiated;
8. If alternatives exist on previous choices impose a new constraint for the cost variable to be less than the cost returned by the LP solution and continue point 9 else end the search;
9. Backtrack to the point 3 and compare the new upper bound (best cost so far) with the cost returned by the LP solver. If the new upper bound is smaller than the cost returned by the LP solver then fail, else remove the value from the variable domain and jump to step 4.

VI. Computational Results

In order to evaluate the different methods and variants we have used a small subset of problem instances created by our research group. Each of the instances includes the general problem parameters as unit characteristics, demand profiles, operational costs, maintenance cost, specific resource and time constraints. The instances were created based on real data provided by the Portuguese Electric Power Generation Company (EDP). Each of the instances varies in terms of search space size. This means that it not only the number of units and periods can vary but also the constraints "intensity". Especifically, an instance is named *msup_n_t_x*, where *n* is the number of units, *t* the number of periods and *x* the intensity of the constraints. *x* can be *[a,b,c,d,e]* where *a* corresponds to the most relaxed constraints and *e* to the most tight. For example, for two instances with *t=24,* in the case where *x=a* the time window *a* can be *[2,...,24]* and for *x=e, [12,...,16]*. Problems of type *e* are therefore more constrained than of type *a*. The idea is to test if the CLP performance increases or not on tight search spaces due to constraints.

All the methods where implemented on ECLiPSe constraint logic programming system and XPRESS-MP LP solver, running on an AMD at 900Mhz, with 256Mb of memory, and running on WINXP. Since none of our methods have stochastic components we only need one run for each instance.

Table I shows the computation time, in seconds, for each of the proposed methods, namely SB, IC1, IC2 and VF for ten MSGU instances. Table II shows the computation time for a pure LP approach and a pure CLP approach. As explained above, the name of each instance indicates a different type of data in size and constraints. (1) and (2) correspond to instances of the same type but with different data.

Table I- Computation Time for each pair MSGU Instance/Method

Instance	SB	IC1	IC2	VF
msh_10_24_a (1)	15,0	25,7	364,4	225,1
msh_10_24_a (2)	0,5	3,3	5,6	4,8
msh_10_24_b (1)	9,5	13,2	5424,2	2299,9
msh_10_24_b (2)	0,2	0,2	4,5	2,2
msh_10_24_c (1)	68,9	64,8	1022,3	851,9
msh_10_24_c (2)	4,6	2,7	23,6	78,4
msh_10_24_d (1)	2,1	0,7	53,2	49,9
msh_10_24_d (2)	53,6	7,6	218,5	638,7
msh_10_24_e (1)	12,6	1,7	24,3	187,6
msh_10_24_e (2)	62,6	8,3	126,6	814

Table II- Performance of a pure CLP approach and pure LP approach

Instance	LP	CLP
msh_10_24_a (1)	22,2	25,7
msh_10_24_a (2)	11,8	6,0
msh_10_24_b (1)	14,1	657,1
msh_10_24_b (2)	9,9	0,2
msh_10_24_c (1)	49,8	745,0
msh_10_24_c (2)	10,5	4,2
msh_10_24_d (1)	24,3	750,0
msh_10_24_d (2)	17,7	923,2
msh_10_24_e (1)	10,9	63,9
msh_10_24_e (2)	14,5	492,3

Despite not being reported here, we registered the number of backtracks. We also registered the number of times a cut occurred in the search tree due to the bounds returned by the LP solver. We named this as "number of LP actions". Analysing the registered data we verify the existence of a correlation between the computation time and the number of backtracks. We also verified a correlation between the computation time and the number of LP actions. However, these two correlations can vary, as expected, from case to case. For example, in the case of the number of LP actions, one cut of the search tree, can include 10 or 100 branches, leading to different computation times.

Regarding Table I, we can see that SB and IC1 methods are the best for the tested MSGU instances. SB shows slight better results for the less constrained instances and poor results for the most constrained. One possible reason is that the variable and value selection heuristic for IC1 is more general and does not take as much advantage of the problem characteristics as the SB selection heuristics. The power of constraints is then more

notorious in the SB method than in the IC1. So, SB has advantage for the most constrained instances. As desired, for this two methods the overhead associated with double modelling and solving is compensated, since the results are better than the ones from the pure CLP and LP approaches.

In the case of VF and IC2 the performance was very poor. In fact, both methods perform even worse than the pure CLP and LP approach.

In the case of VF we have noticed that very rarely the solution returned by LP solver is feasible. Even when almost all the variables are instantiated the solution returned by the LP solver is not integer and so, not feasible. Consequently, the time overhead associated to the solving process and feasibility verification does not bring any advantage, just increasing the computation time. In order to overcome this problem we tested the method maintaining the integrality constraint while relaxing other complex constraints.

Table III- Performance of a pure CLP approach and pure LP approach

Instance	VF (int.)
msh_10_24_a (1)	210,1
msh_10_24_a (2)	8,8
msh_10_24_b (1)	2499,9
msh_10_24_b (2)	12,3
msh_10_24_c (1)	743,4
msh_10_24_c (2)	98,4
msh_10_24_d (1)	59,9
msh_10_24_d (2)	842,1
msh_10_24_e (1)	243,2
msh_10_24_e (2)	912,0

Table III shows the test results. We have verified that the number of feasible solutions greatly increased but the time needed for the LP solving also increased, making the global performance even worse.

In the IC2 case, we believe the selection method almost completely looses problem information. So, the search is almost random. This is one possible reason for the bad performance of the method.

VII. Conclusion

In this work we evaluate the performance of 4 hybrid methods on solving the MSGU problem. The basic idea of each of the methods was to take advantage of the modelling and constraints solving capabilities of CLP while overcoming its limitations by introducing global reasoning capabilities of LP. CLP allows the definition of constraints in a more natural and compact way. The resulting programs are usually small, understandable and therefore very easy to modify and maintain. If CLP controls the search, using local and global constraints, and at the same time is complemented by a global view given by LP, results can be revolutionary.

In the proposed methods we have used incomplete double modelling where the problem model is developed using CLP and then part is converted into linear inequalities suitable to a LP solver. After this, the CLP solver instantiates variables and propagates the constraints while communicating with the LP solver, passing new constraints and receiving tighter bounds and search information. In the end, the overhead associated with the communication and LP solving process is compensated by the search tree reduction in seek for a global optimal solution.

Preliminary tests showed that two of the methods, namely IC1 and SB can do better than pure CLP or LP approaches for the considered MSGU instances. We also verified that the other two proposed methods, IC2 and VF, perform poorly in general, and significantly worse than all the other methods.

Based on the test we can conclude that the hybridisation can compensate, particularly if the right cooperation is implemented between CLP and LP. Despite being possible to find several integration schemes in the literature, we think that the proposed methods can do their jobs well, and at the same time are easy to implement and maintain.

VIII. Future Work

Increasing the number of tests would be the first thing to do. We are particularly interested on testing our methods on larger instances. We also think that more can be done concerning the variable and value selection heuristic, namely in the IC1 case. We believe that more dedicated strategies, exploiting some problem characteristics, can improve the method perfomance.

Finally we are interested in including these methods in a more general search framework as the one named Reduce and Assign [20], developed by our research group.

References

[1] Mark Wallace, S. N. J. S., "ECLiPSe : A Platform for Constraint Logic Programming," London, 1997.

[2] Ottosson G., Hooker J., Thorsteinsson S., and Kim Hak-Jin, "On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization," *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 136-141, 1999.

[3] Beringer H. and Backer B. Combinatorial problem solving in constraint logic programming with cooperating solvers. In: *Logic Programming: Formal Methods and Pratical Applications*, ed. Beierle C. and Plumer L. Elsevier, 1995.pp. 245-272.

[4] Refalo P., "Tight cooperation and its application in piecewise linear optimization," Principles and Practice of Constraint Programming, Alexandria, Virginia, USA, pp. 375-389, 1999.

[5] Heipcke S., An example of integrating constraint programming and mathematical programming 1999.

[6] Robert Rodosek, Wallace Mark, and Hajian M., A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, vol. 86, pp. 63-87, 1999.

[7] Harjunkoski I., J. V. G. I., Hybrid mixed-integer/constraint logic programming strategies for solving scheduling and combinatorial optimization problems. *Computers and Chemical Engineering*, vol. no. 24, pp. 337-343, 2000.

[8] Jain V. and Grossmann I., Algorithms for Hybrid MILP/CP Models for a Class of Optimization Problems. *INFORMS Journal of Computing*, vol. 13, pp. 258-276, 2001.

[9] Hooker J., Ottosson G., and Kim H., A declarative modeling framework that integrates solution methods. *Annals of Operations Research*, vol. pp. 141-161, 2001.

[10] Hooker John N., Ottosson Greger, Thorsteinsson Erlendur S., and Kim Hak-Jin, On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization, *Proceedings of the Sixteenth National Conference on Artificial Intelligence {(AAAI-99)}*, pp. 136--141, 1999.

[11] John Hooker, "Logic-Based Benders Decomposition," PA 15213 , 1995.

[12][Hooker J. N., Ottosson G., Thorsteinsson E. S., and Kim Hak-Jin, A Scheme for Unifying Optimization and Constraint Satisfaction Methods. *Knowledge Engineering Review*, special issue on {AI/OR}, vol. 15, pp. 11-30, 2000.
Notes: Alternate Journal: Knowledge Engineering Review, special issue on {AI/OR}

[13] Hooker J. and Osorio M., Mixed Logical/Linear Programming, *Discrete Applied Mathematics*, vol. 96-97, pp. 395-442, 1999.

[14] Bockmayr A. and Kasper T., Branch-and-Infer: A unifying framework for integer and finite domain constraint programming *INFORMS* J. Computing, vol. 10, pp. 287-300, 1998.

[15] M. Weedy. Electric Power Systems, Chichester: John Wiley Sons, 1992.

[16] El-Sharkh M., Yasser R., and El-Keib A., Optimal Maintenance Scheduling for Power Generation Systems - A Literature Review, *Proceedings of the Maintenance and Reliability Conference MARCON98*, pp. 20.01-20.10, 1998.

[17] Beldiceanu N., Introducing global constraints in CHIP, *Mathematical and Computer Modelling* , vol. 20, pp. 97-123, 1994.

[18] Gomes N. and Vale Z., "Constraint Based Maintenance Scheduling of Electric Power Units ," *Power and Energy Systems (PES2003)*, Palm Springs, California, USA, pp. 55-61, 2003.

[19] G. Christelle Guéret, P. Christian, and Sevaux M. Applications of optimization with Xpress-MP, Dash Optimization, 2002.

[20] Gomes N. and Vale Z., R. C., "Reduce and Assing: A Constraint Logic Programing and Local Search Integration Framework to Solve Combinatorial Search Problems," *Principles and Practice of Constraint Programming - CP 2003*, Ireland, pp. 847-853, 2003.