Combining Metaheuristics and Constraint Programming to Solve a Scheduling Problem

NUNO GOMES, ZITA VALE, CARLOS RAMOS
GECAD – Knowledge Engineering and Decision Support Group,
Institute of Engineering – Polytechnic of Porto, PORTUGAL

Abstract: - In this paper we present a hybrid method, named Quasi Local Search, which combines Simulated Annealing augmented with a type of Tabu List to guide the search globally with Constraint Programming to search locally the optimal solution. The method can also be seen as an integration framework, once the Local Search module is independent of the Constraint Programming one and either can be worked independently. The method is used to solve a Generator Maintenance Scheduling Problem with promising results. We also present a study about the se-lection of certain parameters and neighbourhood structure. The result allows us to conclude about their influence on the method's performance.

Key-Words: - Metaheuristics, Constraint Programming, Hybrid Schemes, Maintenance Scheduling

1 Introduction

From a real world point of view, one of the most appreciated strengths of scheduling and planning tools is flexibility. Particularly, planning and scheduling engineers need tools capable of model the all problem in order to obtain useful solutions. To develop this type of "solving/decision support" tools, Constraint Logic Programming (CLP) is a suitable approach. With CLP the model problem and solving method can easily be developed as well as the corresponding computer program. The latter are usually small and easy to maintain. Evidence of these strengths is the huge number of practical applications of CLP [1]. Despite these Strengths, CLP has also some weaknesses. The most relevant one is some lack of efficiency when solving large and/or few constrained problems. One important reason for this efficiency problem is related to the underlying search mechanism. In fact, within CLP, problems are modelled as a set of variables with domains and a set of constraints restricting the possible combinations of the variables values. Constraints are used actively in order to prune the search space by removing inconsistency values from the variables domain. This action is achieved by a propagation (or filtering) algorithm associated with each constraint. Propagation algorithms are usually incomplete: once propagation is finished, some inconsistent values may remain in the variable domains. In order to find a solution, we need a search method to instantiate all the variables. In one of the most used search strategies, for each variable a value is assigned incrementally until a complete solution is found or a constraint is violated. If a constraint is violated, the last assignment is undone and an alternative value is chosen ("enumeration"). If no value assignment is consistent, the search backtracks to a previously assigned variable, and so on. The result is a depth-first tree search explored using Chronological Backtracking.

Even with very efficient search methods and filtering algorithms capable of pruning significantly the variables domain, it is impractical to explore all the search tree of a real world problem in a depth-first manner. This can prevent the search to find the optimal, or even good quality solutions. In the literature, we can find several methods that try to overcome this problem. Some simply explore the search tree in a different order, possibly discarding some unpromising areas, hopefully in an intelligent way. Other methods try to integrate different techniques, from several areas in order to improve search efficiency. This last type is actually one of the successful research lines within CLP (see section 2). In this work we present a hybrid method that falls in this last research line. The method uses Simulated Annealing (SA) augmented with a type of Tabu List to guide the search globally and CLP to search locally the optimal solution. Basically, CLP can be seen as a good tool to explore the neighbourhood efficiently. We also test the method with a Maintenance Scheduling Problem and study the influence of CLP integration in the SA parameters. Results show that the hybrid method can improve significantly a pure CLP approach.

2 Integration Approaches

The CLP framework allows the integration of techniques from different areas. A recent and successful research line, where this work belongs, is the integration of Local Search (LS) based on methods with CLP. In analyzing the literature we can classify the integration methods in two big groups: CLP is used to find a partial or a complete solution in certain points of the search. Then the solution is completed or improved using some type of LS based method; LS is used to guide the overall search. Then CLP is used to explore the neighbourhood by se-

lecting the right neighbour or removing the unpromising ones. The first group is less common. However, we can find some works on the literature. For example in [2] an hybrid method is used to solve a Workforce Scheduling Problem. For that, they use a CLP based method to pre-schedule the complex tasks (long-duration tasks) and then a SA algorithm is used to schedule the other tasks starting from the pre-schedule. Similarly, in [3], it is presented a new method to solve the Exam Timetabling. The method consists of two phases. In the first phase CLP is used to obtain an initial timetable satisfying all the hard constraints. In the second phase Simulation Annealing is used to improve the quality of the timetable, taking the soft constraints into account.

The second group is larger and different approaches can be found in the literature which are included on it. For example in [4] the authors present an integration framework where they propose to use CLP to solve several sub-problems with a reduced sub-search space. The sub search-space of each sub-problem is selected using a utility function that defines for each pair variable/value a utility value based on the previous search results. With the right utility function it is possible to find the optimal or, at least, a good solution in shorter time. On the same line in [5] the authors use a method named Path-Repair, which merges a Tabu Search together with a filtering technique and conflict-based heuristics to solve an Open Shop Problem. Another example is that of [6] where the authors use a LS method named Large Neighbourhood Search to solving vehicle routing problems. The method explores a large neighbourhood of the current solution by selecting a number of visits to remove from the routing plan, and re-inserting these visits using a constraint-based search.

3 The Integration Method

Due to the exponential complexity of many combinatorial optimization problems, the size of a complete search tree is often intractable. In practice, a search algorithm is only able to explore a small part of a real problem search tree. In case of depth-first search, usually used by CLP methods, only the left part is explored. This can be particularly serious if the first choices are wrong and the search moved apart from the good solutions. Within LS methods there is no exhaustive exploration of the search tree. Alternatively, each solution is obtained interactively through modifications of a previous solution (local move), which can be seen as a jump in the search tree. This mechanism can increase the search speed even if some wrong decisions are made. However, the modification set is not usually unitary, which results in several possible moves and an equal number of solutions. Each of these solutions is called a neighbour, and the whole set, a neighbourhood. Obviously, one crucial aspect of all LS methods is the choice of the neighbourhood structure and its exploration. In real problems with several types of constraints and for which large neighbourhoods are important (see [7]), to validate and to choose the right neighbour can constitute a hard problem itself. In fact it is necessary to model the problem with all its constraints and then use the problem model to determine the best neighbour. The basic idea of the method presented in this paper consists of taking advantage of modelling and pruning capabilities of CLP to solve combinatorial problems, but at the same time using a type of LS method to avoid the exhaustive search tree exploration underlying to the basic CLP search method. From other point of view, the basic idea consists of taking advantage of modelling and pruning capabilities of CLP to explore larger and more complex neighbourhoods increasing the efficiency of traditional LS methods.

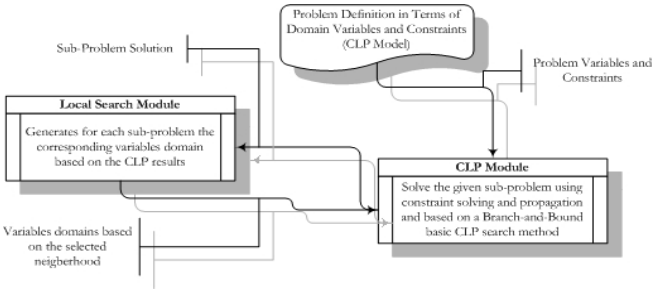

Fig. 1. Integration Scheme

3.1 The General Integration Scheme

The method presented in this paper is based on two modules, one responsible for guiding the global search through the search space and the other responsible for exploring selected search space areas. The first module is based on a LS method and the second on a CLP method. The interaction between the two modules can be seen in figure 1. As we can see, initially the problem model is developed in terms of domain variables and constraints and passed to the CLP module. The CLP module is built over a CLP system and comprises a certain search method, specifically a Refinement-based one and a constrain store with the corresponding filtering algorithms. On each iteration the CLP module is responsible for finding the optimal solution of a given sub-problem. The sub-problem considers all the global problem constraints plus a new set (of the LS responsibility), that limits certain variables domains (reduce the search space). The corresponding sub-search space can be seen as a neighbourhood with valid and non valid solutions. In this way, finding the optimal solution of the sub-problem corresponds to find the best neighbour or to prove that a better neighbour does not exist. Based on the result returned by the CLP module the LS module should define, on each iteration, a new neighbourhood and the corresponding constraint set.

As stated in the previous section, from the CLP point of view, the use of the LS component allows the exploration of the search space in a non exhaustive way. In fact the search is guided by the LS module from one point to another where the CLP module performs a localized exhaustive search. From the LS point of view, the CLP

module allows the exploration of larger and more complex neighbourhoods which increase the chances of success. Another important characteristic of our method is that it is independent of the problem, as of the CLP module or LS module. Naturally, the performance of each of the modules individually is determinant for global method performance

3.2 The CLP Module

As referred above, constraint propagation is not a complete method, so a search scheme is needed in order to instantiate all the variables and obtain a solution. We also referred that it is possible to find several search schemes of different complexity. The hybrid method, we are presenting, is independent of the CLP module, so any search scheme can be used.

For the problem instances presented in this paper we have chosen a simple chronological backtracking algorithm described in section 1, augmented with a type of Branch-and-Bound suitable to CLP based optimization which is well known among the CLP community. This search scheme was already successfully tested on the same problem instances we use in this work, so a comparison can be done (see [8]). On the rest of this paper we will call this algorithm *Basic Backtrack Search (BBS)*. This algorithm contains primarily two choices, namely variable selection and value selection. The order in which variables and values are selected can have a significant impact on search efficiency.

3.3 The LS Module

The main task of the LS Module is to guide the search globally by defining, on each iteration, the neighbourhood to be explored. This guidance is based on an interactive improvement where only the best neighbours are chosen, and neighbourhoods are created starting from the best neighbours. However, this strategy can rapidly lead the search to get trapped on a local minimum. To avoid this problem we use Simulated Annealing (SA) [9] and Tabu Search (TS) [10]. The main idea of the SA algorithm consists of accepting solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The algorithm starts from an initial solution and a certain value of the so called temperature parameter *T*. Then, on each iteration, the selected solution is immediately accepted if its cost is better than the cost of the previous solution, or is accepted with a probability *p* which is generally computed following the Boltzmann distribution. In this case *p* can be evaluated using expression (1)

where *f(s)* corresponds to the cost of the new solution and f (s_o) to the cost of the previous solution. As the search evolves the *T* parameter should decrease in order to the search converge. The *T* value on each iteration is given by the *cooling schedule*. In fact, theoretical results on non homogeneous Markov chains [11] state that under particular conditions on the cooling schedule, the algorithm converges in probability to a global optimum. Unfortunately, cooling schedules which guarantee the convergence to a global optimum have no use in practical applications, because they are too slow.

$$p = e^{\left(-\frac{f(s)-f(s_o)}{T}\right)} \quad (1)$$

3.3.1 SA Cooling Schedule

One of the important issues for the performance of the SA algorithm is the cooling schedule. Although there isn't a perfect schedule for all the applications, a good cooling schedule should guarantee a fair exploration of the search space before cooling down. The difficulty is to find the right trade off between diversification and intensification [12]. In this work we use a cooling schedule based on a geometric law: $T_{k+1}=\alpha.T_k$ where $\alpha \in (0,1)$. With this type of cooling schedule the temperature decay is controlled by the α parameter. This simple schedule has one problem: once it cools down, there is now hypothesis of reheating again. This means that the algorithm will have difficulties in overcome hills in the search landscape. However since the search effort is shared between the SA and CLP there is some guarantees of a straighter search landscape from the SA point of view.

Several works show that α should belong to the interval (0.8,0.99). However, they also indicate that there is a relation among α and the neighbourhood size. In fact, it seems that with large neighbourhoods α cloud take lower values. Note that α should guarantee a good exploration of the search space, nevertheless this objective can also be accomplished with the consideration of large neighbourhoods or with the executions of several iterations at the same temperature. In this work we will present some tests about this.

3.3.2 Search Cycles and Tabu List

Usually, on combinatorial search problems, the neighbourhoods are symmetrical. In this case the occurrence of cycles in the search is possible. This means being $x' \in N(x)$ and $x \in N(x')$, where $N(x)$ represents the neighbourhood of *x*, then the generation sequence *x->x'->x* could happen in three successive iterations. In the conventional SA the generation probability has a uniform distribution and the probability is inversely proportional to the neighbourhood size. Being *n* the neighbourhood size, then the generation probability P_G of each neighbour can be determined by the expression (2).

$$P_G(i) = 1/n, \forall i \in N(j) \quad (2)$$

Considering (2) we can calculate the generation P^T_G and acceptation P^T_A probability of *j* after *i* respectively using expressions (3) and (4).

$$P^T_G(j->i) = 1/n^2 \cdot P^T_A(i), \forall i \in N(j), \forall j \in N(i) \quad (3)$$

$$P^T_A(j->i) = 1/n^2 \cdot P^T_A(i) \cdot P^T_A(j), \forall i \in N(j), \forall j \in N(i) \quad (4)$$

Where T is the control parameter which in our case is the temperature. Note that the acceptation probability of SA is given by (5).

$$P_A^T(i) = e^{-\left(\frac{f(i)-f(s)}{t}\right)}, \forall i \in S \quad (5)$$

Following we can determine the probability of occurring a cycle in three consecutives iterations.

$$P_A^T(i->j->i) = 1/n^3 \cdot \left(P_A^T(i)\right)^2 \cdot P_A^T(j), \forall i \in N(j), \forall j \in N(i) \quad (6)$$

In our method the neighbour returned by the CLP module is always the same and the optimal, considering that the underlying search scheme is deterministic and complete. The generation probability in this case is given by the expression (7).

$$P_G(i) = \begin{cases} 1 & \forall j \in N(k), f(i) \le f(j), i \in N(k), i \ne j \\ 0 & \exists j \in N(k) \mid f(j) \le f(i), i \in N(k), i \ne j \end{cases} \quad (7)$$

Considering (6) and (7) we can determine the probability of occurring a cycle in three consecutives iterations for our method.

$$P_A^T(i->j->i) = P_j^{Min}(i) \cdot \left(P_A^T(i)\right)^2 \cdot P_A^T(j), \forall i \in N(j), \forall j \in N(i) \quad (8)$$

Where $P_j^{Min}(i)$ corresponds to the probability of i being the best neighbour of j. Following the same reasoning it is possible to generalize expression (6) and (8) to cycles with any size. This means that, comparing (6) and (8), considering n a big number and that $P_j^{Min}(i) >> 1/n$ we can conclude that the probability of occurring cycles in our method is much higher than in conventional SA. This was verified in our tests. In order to overcome this problem, the QLS method includes a type of short term memory analog to the one of the Tabu Search method [10]. The memory mechanism is implemented through a Tabu List. This list keeps the last selected neighbours (solutions) as tabu. On each iteration the tabu neighbours are passed as constraints to the CLP module avoiding their reselection, and consequently cycles. Naturally, the size of the avoided cycles is related to the size of the tabu list. Considering all the above we can finally present the QLS algorithm in Fig. 2.

4 QLS applied to the Generation maintenance scheduling

In order to test the QLS method, we have used a set of the Power Systems Generator Maintenance Scheduling problem (GMS) instances. This 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). The GMS can be a complex problem. Usually it has associated a large and complex set of problem constraints. In this work we consider the following ones: *Resources; Time; Precedence; Capacity; Demand*.

A real GMS 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 GMS problem objective function is a sum of cost terms that shall be minimized. These terms are usually the total Production Costs (PC) and total Maintenance Cost (MC) for the schedule horizon.

```
Develop the problem model in terms of domain variables
and constraints (suitable to a CLP system)
Determine the initial solution using the CLP module
without optimization
Set the cooling schedule (initial temperature T and
number of iterations per temperature NI)
Initialize Tabu List
while stop criteria not reached
   while NI>0
      Impose the needed constraints in order to restrict
      the variables domains according to the neighbour-
      hood defined by the respective neighbourhood func-
      tion.
      Impose the needed constraints to avoid the tabu
      solutions
      Call the CLP module to find the best neighbour s
      If f(s) better than f(s0) then
         s0=s
      else
         s0=s with probability p=e^-(f(s)-f(s0))/T
      update the tabu list if s accepted
   end while
   update T according the cooling function
end while
```

Fig. 1. QLS Algorithm

4.1 General Problem Model

The problem variables are: S_i wich corresponds to the starting maintenance period of unit I; P_{it} which corresponds to power production 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 naturally arise from the problem definition:

$$1 \le S_i - r_i \le T \qquad \forall i \in U \qquad (9)$$

$$e_i \le S_i \le l_i \qquad \forall i \in U \qquad (10)$$

$$P_{it} \le k_i \qquad \forall i \in U, \forall t \in T \qquad (11)$$

$$\sum_i P_{it} \ge d_t \qquad \forall t \in T \qquad (12)$$

$$\sum_i E_{it} \le ar_t \qquad \forall t \in T \qquad (13)$$

$$S_i + r_i \le S_j \vee S_j + r_j \le S_i \qquad (i,j) \in I \qquad (14)$$

$$S_i + r_i \le S_j \qquad (i,j) \in P_r \qquad (15)$$

Where T and U are respectively the available maintenance periods and units. ar_t the maximum number of units which can be maintained simultaneously in period t. k_i Maximum power production capacity of unit i and d_t the demand in period t. I Set of pairs of units which cannot be maintained simultaneously, and Pr set of pairs that have precedence requirements Considering constraint classification from the previous section, we can say that: (9) and (10) are time constraints defining the valid maintenance period; (11) is the demand constraint; (12) is the capacity constraint; Finally, (13), (14) and (15) are resource constraints. Our objective function is given by expression (16) where the goal is to minimize the total maintenance and production costs.

$$\min\left(\sum_i \sum_t c_{it} \times P_{it} + \sum_i Mc_i\right) \quad (16)$$

4.2 Implementation Issues

As referred in section 2.3 we will use the BBS algorithm of **Erro! A origem da referência não foi encontrada.** augmented with a type of Branch-and-Bound as the search algorithm for the CLP module. More efficient methods could be used, as for example that of [13], but for our purposes a complex method could mislead the evaluation of the QLS method. The algorithm performance strongly depends on the variable and value selection strategies, so care should be taken on their choice. In our test we use the *Smallest* variable selection (SVA) heuristic and *Smallest* value (SVU) selection heuristic has defined in [8]. Where SVA selects the maintenance cost variable (Mc_i) with the smallest value in the domain, and SVU selects the smallest value in the domain of the variable (smallest maintenance cost). Obviously, branching is made over the Mc_i variables.

Besides the cooling schedule parameters, another important issue of the LS module is the neighbourhood structure. The GMS problem is globally a scheduling problem. This way, similar neighbourhood structures can be used. One common neighbourhood consists on the permutation of two tasks, although it can be generalized to any number of tasks. Other neighbourhood consists of shifting in time one or more tasks. Within CLP all these neighbourhoods can be easily implemented and possibly more than one at the same time. For example, to implement a structure that shifts the scheduling period to the right, we just need to constraint each variable domain to a set that includes the current solution value and the value plus one. Note, however, that with such variables domain (two values for each variable domain) other neighbourhoods are explored at the same time. One that is tested is task pair permutation. In our tests, some of this neighbourhoods are implemented using a simple strategy which restricts the domain of *n-nfv* (*n* is the total number of variables and *nvf* is a number to define) variables to a single value that corresponds to the value of the last solution.

4.3 Computational Results

In order to evaluate the QLS method performance, well as the influence of each one of its parameters, we realized a large number of tests over several GMS instances. Each instance distinguishes by its size or by the number and complexity of its constraints. The terminology to identify the instance is GMS_NT_NU_T, where: NT is the number of scheduling periods; NU is the number of units to be schedule; and T the problem class. If NT and NU are self explanatory and define the instance size, T is less obvious. T defines the problem class and can take 5 different values, from *a* to *e*. Each value indicates a problem type with different number and complexity of constraints, being class *a* the easier and *e* the difficult to solve. The idea of having different classes of problems is to test the capabilities of the QLS method to solve problems with different difficulty levels relative to the complexity of the constraints. Note that CLP methods are very useful to solve problems with few solutions by opposition to LS methods that perform better in large search spaces with a lot of solutions. All the tests were made in 1,8 MHz Pentium PC, with 512Mb of memory, running Windows XP. Due to the stochastic component of the method all the presented results correspond to the average of 10 runs.

4.3.1 Cooling schedule parameters

As referred in section 2.4.1 we use a cooling schedule based on a geometric function of type $T_{k+1}=\alpha.T_k$. This schedule requires the definition of the initial temperature T_0 and of the cooling rate α. Furthermore, in our case, as in most of the cases, we use a fast cooling rate but considerer several iterations (NI) at the same temperature. Naturally, the number of iterations is related to the cooling rate and if one increases the other should decrease according.

Fig. 3 shows the results obtained with different parameters values for 6 different instances. Each bar graph corresponds to best solution cost found in a limited period of time. However, in order to facilitate the comparison the cost is given in percentage of the worst cost. This means that the worst solution found receives 100% and the others are given in accordance. Note that each colour corresponds to a different cooling rate α and that NI corresponds to the number of iterations at the same temperature. The initial temperature was determined experimentally so that the initial acceptance probability could be near 1. For the remaining parameters we used a tabu list of size 10 and the neighbourhood presented in section 4.2 with *nvf* equal to 1. In analyzing the results we can say that the best cooling rate is around 0.95 for all the tested instances. Furthermore, for values within the interval [0.93,0.97] we can expect a variation inferior to 20% in the results. As expected, the NI value is determinant for the performance and should increase with the size of the instance. Similarly to α a deviation of NI from the best value inferior to 50% produce a variation of the results inferior to 20%. Due to space restrictions we do not show in this paper neither the results for the other tested instances neither for different Tabu List sizes.

4.3.2 Neighbourhood Size

As referred in section 4.2 the neighbourhood, used on these tests, is created by restricting the domain of, n-nvf randomly selected Mc_i variables, to a single value that corresponds to the value each variable had in the last solution. Note that in case the LS module rejects a neighbour an alternative neighbourhood can be created by selecting other nvf variables.

The neighbourhood size is directly related with the *nvf* value. In order to test the influence of the neighbourhood size we have made some tests with *nvf* values varying from 1 to 5. The results were similar for *nvf*=1 and *nvf*=2 being the bests among all. For higher *nvf* values the results are degraded. We believe that a different neighbourhood structure could produce different results with the respect to the *nvf* value. This issue remains for future work.

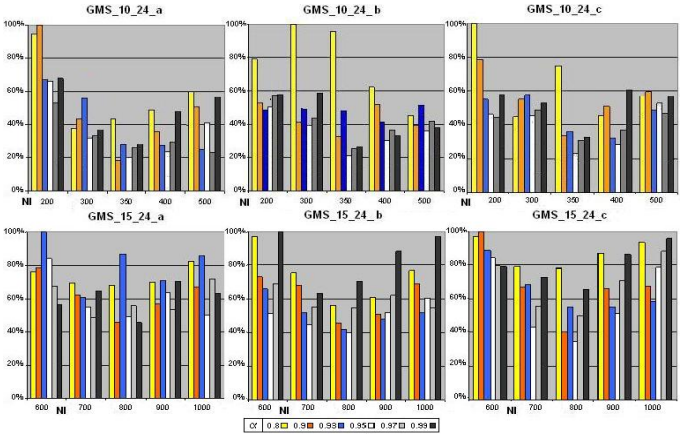

Fig. 2. QLS results for 6 different GMS instances

4.3.3 Comparison with other methods

Table 1 shows the time needed to find the optimal solution of 5 GMS instances by a pure CLP approach and the QLS method. The pure CLP corresponds to BBS method augmented with the Branch and Bound as used in the CLP module. In fact the QLS method can be a pure CLP approach if none restriction is made to the variables domains (the neighbourhood corresponds to the all search space). As we can see the QLS method improves significantly the pure CLP approach. Four some approaches the computation time is reduced by almost 50%. We believe that the results can even be better for instances with larger size.

Table 1. Time needed in CPU seconds to find the optimal solution by the BBS and the QLS method for 5 GMS instances

Instance	BBS	QLS
GMS_10_24_a	35,3	33,2
GMS_10_24_b	153,2	127,1
GMS_10_24_c	242,1	172,0
GMS_10_24_d	260,0	134,3
GMS_10_24_e	93,2	68,7

5 Conclusions and Future Work

In this paper we presented a hybrid method, named Quasi Local Search, which uses Simulated Annealing (SA) augmented with a type of Tabu List to guide the search globally and CLP to search locally the optimal solution. The method can also be seen as an integration framework, once the LS module is independent of the CLP one and either can be worked independently. Results of the application of the QLS method to some Generator Scheduling Maintenance instances show that the method can improve significantly a pure CLP approach. As other SA based methods, the QLS requires the set of certain parameters, namely that of the Cooling Schedule. Results shown that the empirical indications found in the literature can also be used to set the QLS parameters. This conclusion suggests that more successful cooling schedules can be used for the LS module. This remains an open issue for future work. Another important issue of any Local Search based method is the neighbourhood structure. The QLS method allows the simultaneous exploration of several traditional problem specific neighbourhoods. In this work we have shown the results for a simple neighbourhood creation mechanism that consist of restricting the domain of all the variables, except a certain number, to a single value that corresponds to the value the variable had in the last solution. Results show that with this mechanism, the useful neighbourhood size is limited. For future work we want to try other mechanisms. Some work has been done over this issue with promising results.

References:

[1] Francesca R. et al., "Constraint Logic Programming," *ERCIM Net workshop on constraints,* 2000.

[2] Voudouris C., et. al., Solving Large Industrial Problems using Heuristic Search and Constraint Programming, *UNICOM Seminar on Modern Heuristics for Decision Support,* 1998.

[3] Duong T. and Lam K., Combining Constraint Programming and Simulated Annealing on University Exam Timetabling, *RIVF'04,* pp. 205-210, 2004.

[4] Gomes N., Vale Z., and Ramos C., "Reduce and Assign: A CLP and Local Search Integration Framework to Solve Combinatorial Search Programs," *Principles and Practice of Constraint Programming*, 2003.

[5] Jussien N., L. O., Local Search with Constraint Propagation and Conflict-Based Heuristics, *AAAI*, 2000.

[6] Shaw P., Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems," *Principles and Practice of Const. Programming,* 1998.

[7] Waterman M., Neighborhood size in the Simulated Annealing Algorithm, *American Journal of Mathematical and Management Sciences*, vol. 8, 1998.

[8] Gomes N., Vale Z., Constraint Based Maintenance Scheduling of Electric Power Units , *7th IASTED Conference on Power and Energy Systems,* pp.55-61, 2003.

[9] Kirkpatrick S. et. al., Optimization by Simulated Annealing, *Science*, vol. 220, no. 4598, 1983.

[10] Glover Fred, Tabu Search, part I *ORSA Journal on Computing*, vol. 1, pp. 190-206, 1989.

[11] Aarts E., et. al., *Simulated Annealing, Local Search in Combinatorial Optimization*, John Wiley, 1997.

[12] Abramson D. and Dang H., Simulated Annealing Cooling Schedules for the School Timetabling Problem, *Asia-Pacific Journal of OR*, vol. 16, 1999.

[13] Gomes N., et. al., "Hybrid Methods for the Maintenance Scheduling of Generating Units Problems," *ICKEDS´04,* Porto, Portugal , 2004.