Solving Constraint Optimization Problems in Anytime Contexts

Samir Loudni
Ecole des Mines de Nantes
F-44307 Nantes Cedex 3 – France
samir.loudni@emn.fr

Patrice Boizumault
GREYC, CNRS UMR 6072, Université de Caen
F-14032 Caen Cedex – France
patrice.boizumault@info.unicaen.fr

Abstract

This paper presents a new hybrid method for solving constraint optimization problems in *anytime* contexts. Discrete optimization problems are modelled as *Valued* CSP. Our method (VNS/LDS+CP) combines a Variable Neighborhood Search and Limited Discrepancy Search with Constraint Propagation to efficiently guide the search. Experiments on the CELAR benchmarks demonstrate significant improvements over other competing methods. VNS/LDS+CP has been successfully applied to solve a real-life *anytime* resource allocation problem in computer networks.

1 Introduction

It is today indisputable that Constraint Programming (CP) fulfills industrial needs of optimization **off-line**. Many systems which use this technology are already operational in different fields of activity, such as planning, scheduling and resource allocation. In **on-line** optimization context, problems dynamically change according to the evolution of the external environment, and their resolution must respect strong *temporal constraints* (*anytime* contexts). However, CP is not yet adapted for solving such *on-line* problems : no guarantee is supplied on the execution times, and current CP solvers do not lend themselves to the combination with other search methods which seem better adapted to respect temporal constraints, such as *stochastic search methods*.

Our objective is to conceive *anytime* algorithms based on CP and to guarantee the following main properties : to produce in a very short time solutions of good quality, and more importantly to improve them gradually as computation time increases. To our knowledge, our proposal is one of the first attempt to efficiently address the problem of building *anytime* algorithms with Constraint Programming.

This paper presents a new hybrid method, called VNS/LDS+CP, dedicated to the efficient computation of high quality solutions (possibly suboptimal) in an *anytime* context. Our method combines a Variable Neighborhood Search [Hansen and Mladenovic, 1997] (it performs moves like local search), and a Limited Discrepancy Search [Harvey and Ginsberg, 1995] with Constraint Propagation to efficiently guide the search.

Section 2 describes the general context in which we take place. Section 3 reviews the hybridization of search algorithms and justifies our first choices. Section 4 details our proposal (VNS/LDS+CP). Section 5, gives computational experiments on CELAR benchmarks ; both mechanisms of VNS/LDS+CP are then finely studied. Section 6 shows how VNS/LDS+CP has been successfully applied to solve a difficult network problem. Finally, we conclude and draw a few perspectives.

2 Anytime constraint optimization context

2.1 Anytime algorithms

The term *anytime* algorithm was coined by Dean and Boddy in the mid-1980s in the context of their work on time-dependent planning [Boddy and Dean, 1994]. Contrary to a standard algorithm, where no result is available until its ending, an *anytime* algorithm can be stopped, at any time, to provide a solution of increasing quality over time.

Performance profiles describe the evolution of the solution quality (produced by the algorithm) as a function of the computation time. They provide a more precise description of the performances and of the behavior of an algorithm than the best known solution reported by *usual algorithms*. In the sequel of this paper, we consider *mean performance profiles* : such curves are built empirically by collecting statistics on the performance of the algorithm over many input instances.

There are two kinds of *anytime* algorithms, namely, *interruptible* and *contract* algorithms [Zilberstein, 1996]. A contract algorithm requires that the computing time must be known prior to its activation. It must produce a solution within a contract of time, and if it is interrupted before the end of the contract, there is no guarantee about the quality of the solution. However, for an interruptible algorithm, no information is (a priory) available about the allowed computation time, and a solution may be asked for at any time. The algorithm must always provide a *good* solution, and more importantly, continue to refine it.

In this paper we are interested by the design of interruptible algorithms, since every interruptible algorithm is trivially a contract algorithm, but the converse is not so immediate.

2.2 Valued Constraint Satisfaction Problems

The Valued CSP (VCSP) [Bistarelli *et al.*, 1999; Schiex *et al.*, 1995] framework is an extension of the CSP (*Constraint Satis-*

faction Problem) framework, which allows *over-constrained* problems or preferences between solutions to be dealt with.

More formally, whereas a CSP is defined as a triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V}$ is a set of *variables*, $\mathcal{D}$ a set of finite *domains* associated to the variables and $\mathcal{C}$ a set of *constraints* between the variables, a VCSP can be defined as a CSP provided with a valuation structure $\mathcal{S}$ and a valuation function φ. The valuation structure $\mathcal{S}$ is a triplet $(E, \succ, \otimes)$, where E is a valuation set, $\succ$ a total order on E, $\top$ and $\bot$ the maximum and the minimum element in E and $\otimes$ a binary operation closed on E. The valuation function φ, defined from $\mathcal{C}$ to E, associates a valuation to each constraint; the valuation of a constraint denotes its importance.

Let $\mathcal{A}$ be an assignment of all the variables and $\mathcal{C}_{unsat}(\mathcal{A})$ be the set of the constraints unsatisfied by $\mathcal{A}$. The valuation of $\mathcal{A}$ is the aggregation of the valuation of all the constraints in $\mathcal{C}_{unsat}(\mathcal{A}) : \varphi(\mathcal{A}) = \otimes\ \varphi(c)$ for $c \in \mathcal{C}_{unsat}(\mathcal{A})$.

The objective is to find a complete assignment with minimum valuation. Specific frameworks depend on the retained operator $\otimes$: Classical CSP ($\wedge$); Possibilistic CSP (max); Lexicographic CSP ($\cup$ on multi-sets); Weighted CSP (Σ). As many real-life problems use an additive criterion, we only consider weighted CSP (WCSP) in the sequel of this paper. From an algorithmic point of view, WCSPs are generally the most difficult to solve [Schiex *et al.*, 1995].

3 Hybridizations of algorithms

3.1 Complete search vs local search

Exact (or complete) tree search methods, such as *Branch and Bound*, are able to produce both an optimal solution, and a proof of optimality. But, because of their exponential worst-case behavior, they may be extremely time consuming. Moreover, it has been experimentally observed that, due to their systematic way of exploring the search space, the quality of their intermediate solutions is usually very poor.

Due to their opportunistic way of exploring the search space, approximate (or incomplete) methods, based on *stochastic local search* (as *Simulated Annealing*, *Tabu Search*), can provide good solutions within a reasonable computing time. But, such methods do not guide fast enough the search towards the best neighbor solutions. Indeed, they may waste a lot of time trying to improve the current solution with no success ; this is the case when they remain blocked in *local minima* during a prohibitive time. Such situation is not suitable in an *anytime* context, since the quality of solutions has to be improved gradually as fast as possible, as the computing time increases. Moreover, pure local search algorithms generally require a lot of time to adjust their *noise* parameters ; their efficiency strongly depends on the value of these parameters, which are generally application-dependent.

3.2 Hybrid algorithms

Hybrid algorithms [Focacci *et al.*,] provide appropriate compromises between both kinds of search. More precisely, they are very efficient by combining the advantages of both *constraint propagation* (complete search) and *opportunistic exploration* of the search space (local search).

Algorithm 1: The general VNS/LDS+CP algorithm

```
   VNS (pi : Problem, Size, discrep : Integer, Str : Strategy )
   begin
       size ← Size
       s* ← InitialAssignment (pi)
       s ← s*
       ub ← Cost (s*)
       for i ← 1 to MAXMOVES do
1          rel ← Relax (s,size,Str)            // relax solution
2          s ← Rebuild (s,rel,discrep)         // rebuild solution
           if Cost (s) < Cost (s*) then
               s* ← s
3          size ← ControlNeighborhoodSize (size)
       return s*
   end
```

Intertwined hybridizations are the most attractive and relevant hybridizations of algorithms where both complete and local search are closely mingled during computation. The first kind of such hybrid algorithms belongs to the family of local search methods and uses ideas from CP in order to make *large neighborhoods* (LNS [Shaw, 1998]) more tractable. A second kind belongs to the family of exact search and uses some local search principles to improve the partial solutions at each node of the search tree [Prestwich, 2000] or to explore a set of solutions close to the greedy path in a search tree [Caseau *et al.*, 1999].

3.3 Hybrid algorithms for anytime contexts

VNS/LDS+CP is an *Intertwined hybridization* algorithm. From local search, we have retained a *Variable Neighborhood Search* (VNS) which extends the principle of large neighborhoods (LNS) by *dynamically adjusting* the neighborhood size, when the current solution is a *local optimum*. This choice will partially remedy to the weaknesses of pure local search methods. Indeed, the more the neighborhood is potentially large the more there are chances that it contains good solutions and thus improves quickly the current solution. However, as neighborhoods grow larger, finding the best neighbor may require a too expensive computational effort. So, we have selected the LDS partial search combined with constraint propagation (lower bounds computation) to efficiently explore these neighborhoods.

4 The VNS/LDS+CP Method

VNS/LDS+CP is basically a local search method, as depicted in algorithm 1. It differs from classic local search methods by the size of the neighborhoods, which is adjusted dynamically during the search. It starts with an initial complete assignment s^*; then, at each move, it *relaxes* (or unassigns variables) a large part of the current solution s and then *rebuilds* it (reassigns variables) by selecting the best neighbor that strictly improves the cost of the current solution. The algorithm ends when the maximum number of local moves (MAXMOVES) is reached.

LDS explores the neighborhood defined by the relaxed part of the solution. It benefits from constraint propagation based on *lower bounds* computation, and on *dynamic heuristics* for variable and value ordering. Moreover, only judicious neighborhoods, related to *conflict variables* (i.e. variables occur-

Algorithm 2: Relaxing and rebuilding a solution

```
Relax (s : Solution, size : Integer, Str : Strategy)
begin
1   ν ← GenerateNeighborhood (s,Str)
    rel ← φ
    while | rel | ≠ size and ν ≠ ∅ do
2       x ← RandomElement (ν)
        rel ← rel ∪ {x}
        ν ← ν \ {x}
    return rel
end
Rebuild (s : Solution, rel : VarSet, discrep : Integer)
begin
    if | rel | = 0 then
        if Cost (s) ≺ ub then
            ub ← Cost (s)
            BestSolution ← s
    else
        x_i ← ChooseVariables (rel)
        D_{x_i} ← SortDomain (x_i)
        k ← 0
        for j ← 1 à DomainSize(x_i) and k ≤ discrep do
            s[i] ← D_{x_i}[j]
            lb ← LB(s,x_i)
            if lb ≺ ub then
                Rebuild (s, rel \ {x_i}, discrep − k)
            k ← k + 1
    return BestSolution
end
```

ing in *violated constraints*), are considered. Such a choice prevents LDS from modifying the value of conflict variables.

4.1 Relaxing a solution

Algorithm 2 describes how to select the variables to be relaxed. The function of line (1) computes the set ν of current variables candidate to relaxation, according to a strategy Str. A basic strategy is to select all the variables that are in conflict. $size$ elements of ν are randomly chosen to constitute rel the set of variables to be relaxed. This choice enables a balance between choosing variables according to a specific strategy and completely at random. Experiments have shown that the introduction of some randomness enables the search to escape quickly from *local minima*.

4.2 Control of the Neighborhood Size

Initially, the neighborhood size ($size$) takes a *minimum value*. To control the neighborhood size, different strategies have been implemented. The best strategy we have found, increases systematically $size$ by one, each time the method does not improve the cost of the current solution.

4.3 Rebuilding a solution

Algorithm 2 defines the function `Rebuild`. The global variables ub and lb record the upper and lower bounds of the problem optimum. LB(s,x_i) computes a lower bound of the subproblem, limited to the variables after i (i included). $discrep$ sets the maximum number of choices that we can diverge from the heuristic (*discrepancies*).

Limited Discrepancy Search

The idea of LDS [Harvey and Ginsberg, 1995] is to explore heuristically good solutions that might be at a limited distance from *greedy solution*. LDS ensures a more balanced exploration of the search tree, and speeds up the reconstruction step, thus improving the performance profile of our method. LDS starts from the solution computed by the value heuristic, and successively explores solutions by increasing the number of discrepancies, until the fixed maximal number of discrepancies is reached.

We have used a generalized version of LDS for n-ary trees. The discrepancy is measured according to the rank of the value chosen for every variable in the order given by the heuristic on values. We count a single discrepancy for the *second cheapest* value of one variable, two discrepancies for either the *third cheapest* value of one variable, or the second cheapest value of two variables, and so on. We only perform *one iteration* of LDS, for a fixed discrepancy. This prevents re-visiting leaf nodes.

Constraint Propagation

One of the main strengths of our approach lies in the use of constraint propagation to prune useless sub-trees and to guide the choice of values during the reconstruction, while keeping this step fast enough. At each node of the search tree, *lower bounds* are computed in order to locally exclude all partial solutions which cannot lead to complete assignments of better valuation than the current best solution. To our knowledge, no method based on large neighborhoods (in particular VNS) uses such a mechanism.

To compute lower bounds, we have adapted the following algorithms [Larrosa *et al.*, 1999] for WCSPs : (i) *Partial Forward Checking* and *Directed Arc Consistency* (PFC-DAC), all the constraints are directed, and DAC are computed from a *directed constraint graph* (we have used specific data-structures to relax the condition of static variable ordering on DAC) ; (ii) *Reversible DAC* (PFC-RDAC), constraints directions can change dynamically during search, looking for a good directed graph causing a high lower bound ; (iii) *Maintaining Reversible DAC* (PFC-MRDAC), DAC are maintained during search, propagating the effect of value pruning.

Variables and Values heuristics

Our heuristic for variable ordering first selects the variable having the lowest ratio *domain cardinality divided by future degree*. Constraint propagation, based on Forward Checking + Directed Arc Consistency, allows us to use a *dynamic minimum inconsistency count* value ordering. During search, for each value, so-called *Inconsistency Counts* (*ic*) and *Directed Arc Consistency counts* (*dac*) which record the look-ahead effects on future (unassigned) variables, are computed. Values are selected by their increasing *ic* + *dac*.

5 Experimentations on CELAR Benchmarks

5.1 Instances of CELAR

The CELAR (French *Centre d'Électronique de l'Armement*) has made available a set of 11 instances of the *Radio Link Frequency Assignment Problem* (RLFAP) [Cabon *et al.*, 1999]. Most of them can be naturally cast as binary WCSPs. For our experiments, we have selected instance 6 (which involves 200 variables, having an average of 40 values in their domain, and 1322 constraints), instance 7 (which involves 400 variables,

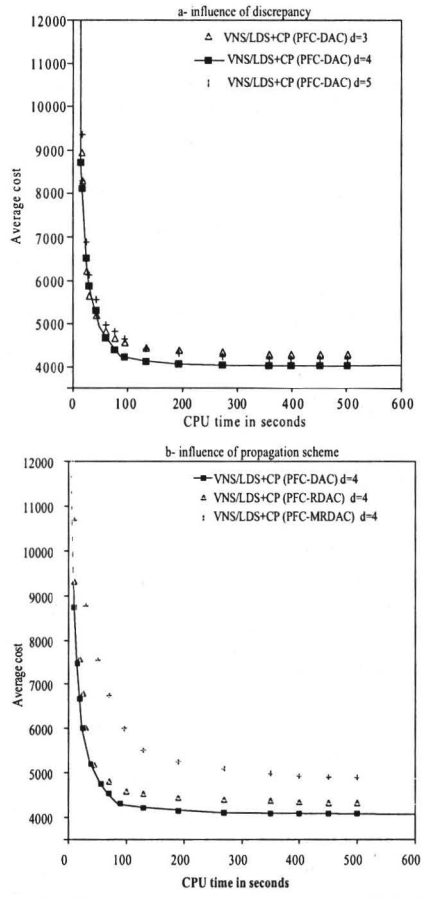

Figure 1: Performance profiles for VNS/LDS+CP, for different setting parameters, on instance 6.

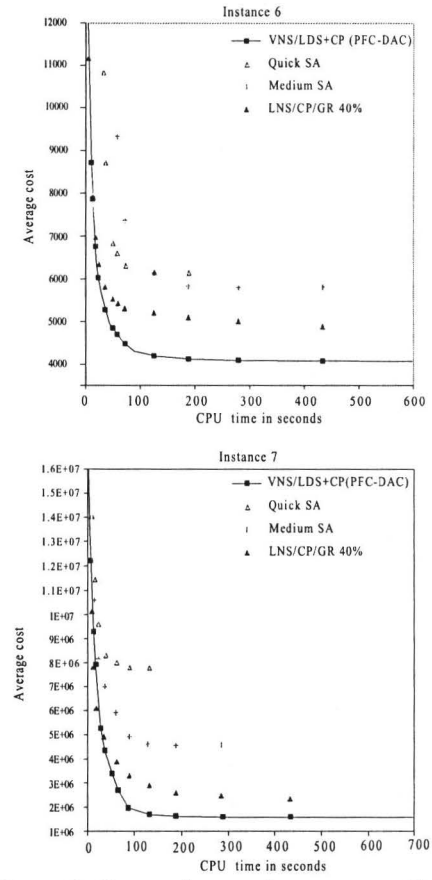

Figure 2: Comparing performance profiles.

having an average of 40 values in their domain, and 2866 constraints), and instance 6-$Sub1$ (which involves 28 variables, having an average of 40 values in their domain, and 314 constraints) extracted from instance 6. These instances (6 and 7) are the most difficult ones. Whereas the optimum of instances 6 and 6-$Sub1$ are known (respectively 3389 and 2669), the optimum of instance 7 is still unknown; the best known upper bound is equal to 343592.

5.2 Experimental Methodology

Our method has 4 parameters : the maximum number of local moves (MAXMOVES), the initial neighborhood size ($size$), the number of discrepancies ($discrep$), and the propagation scheme. For all experiments, MAXMOVES has been set to 150 and $size$ to 4 ; it is the best value we have found among the following set of values : $\{2, 3, 4, 5, 6\}$. We carried out experiments with different values of discrepancies ($discrep \in \{2, 3, 4, 5\}$) and with three different propagation mechanisms : PFC-DAC, PFC-RDAC and PFC-MRDAC.

For each combination of parameter settings, we ran VNS/LDS+CP 50 times on the considered instances. During each run, the best cost of the current solution has been recorded at regular time intervals (10 seconds). All plotted figures are *mean performance profiles* over the 50 runs. The methods have been implemented in choco [Laburthe, 2000].

5.3 Setting parameters for VNS/LDS+CP

Number of discrepancies

Figure 1 (top) shows the *performance profiles* obtained with PFC-DAC, for different settings of $discrep$ on instance 6. Performance profiles for $discrep = 2$ and 3 are almost the same. We give only the curve associated to $discrep = 3$. Results indicate, as expected, that the number of discrepancies has a great impact on the quality of the computed solutions. LDS with ($discrep = 4$) gives the best results. The poor results for ($discrep > 4$) are probably due to the fact that our value ordering does not make so many mistakes. Indeed, we exploit the propagation scheme to guide the value choices. Thus, few discrepancies are necessary to repair heuristic errors. In contrast, for low discrepancies ($discrep \leq 3$), the value heuristic too closely limits the solutions we can find, and so, it provides poor guidance to good solutions. For the instance 7, $discrep = 4$ is also the best value.

Propagation mechanism

We turn now to choose the propagation scheme which produces significant lower bounds and, as global effect, leads to a better behavior of our method. Figure 1 (bottom) reports the *performance profiles* obtained for the three propagation algorithms, with $discrep = 4$ on instance 6. LDS with PFC-DAC leads to very substantial profits. Because of the overhead of propagation over small neighborhoods these benefits decrease clearly when using stronger propagation (PFC-

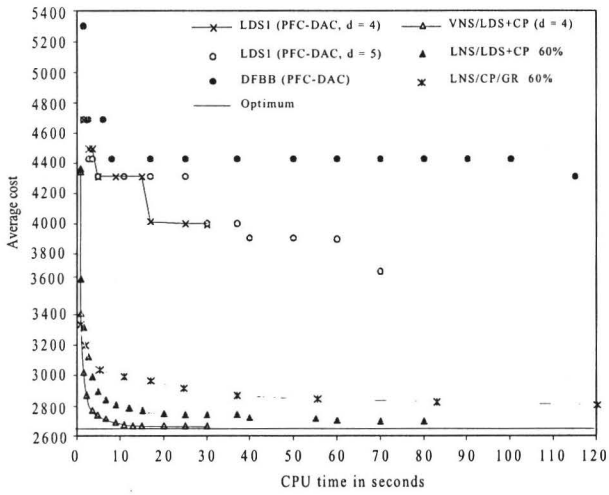

Figure 3: Comparing performance profiles on 6-$Sub1$.

RDAC and PFC-MRDAC). Indeed, compared to PFC-DAC, the two other propagation schemes perform more work per node to finally obtain results very close to PFC-DAC. So, the more complex and time-consuming propagation schemes do not pay-off for this context.

5.4 Comparisons and discussion

We have compared VNS/LDS+CP ($discrep$ = 4 with PFC-DAC) with LNS/CP/GR [Lobjois *et al.*, 2000], another hybrid method which also relies on solving VCSPs in an interruptible context, and also with two standard versions of *Simulated-Annealing* (SA) : *Quick* and *Medium*.

LNS/CP/GR is based on the principle of LNS with neighborhood size being *constant* during all the search. To rebuild the relaxed variables, it uses a *greedy algorithm* combined with constraint propagation. But this propagation (performed variable per variable) is only used for selecting the best values for each variable, which is fundamentally different from our CP based on usual propagation for WCSP. For instances 6 and 7, the percentage of unassigned variables represents 40% of the total number of variables [Lobjois *et al.*, 2000].

Figure 2 compares the performances of the four methods. The *performance profile* of VNS/LDS+CP is always better than those of the two versions of *SA*. At the beginning, the profile of VNS/LDS+CP is very close to that of LNS/CP/GR. But after few seconds of computation ($\leq$ 20 seconds on instance 6 and $\leq$ 35 seconds on instance 7), VNS/LDS+CP becomes very effective and always provides solutions of better quality, thus clearly outperforming LNS/CP/GR on both instances. This behavior can be explained by the fact that, at the beginning, the size of the neighborhood is relatively small and consequently the benefits of constraint propagation are poor. But as soon as this size becomes sufficiently large, our propagation mechanism (lower bounds computation) produces more pruning, and becomes very effective. This leads to a better compromise between quality and time for VNS/LDS+CP. This confirms our choice for computing significant lower bounds. Note however, the good *anytime* behavior of LNS/CP/GR compared with those of *SA*.

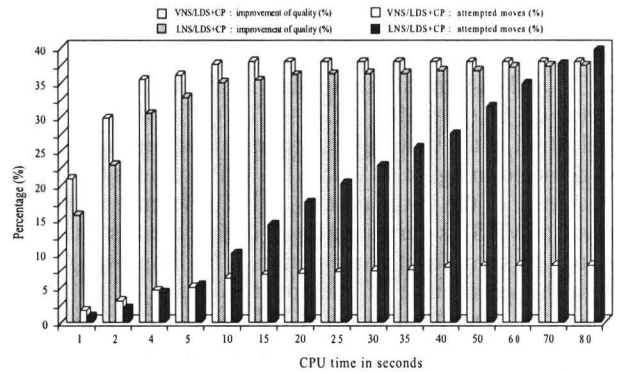

Figure 4: Comparing VNS and LNS on instance 6-$Sub1$.

5.5 Study of mechanisms VNS and LDS+CP

In order to evaluate the contribution of each component of VNS/LDS+CP we have carried out experiments on instance 6-$Sub1$, with MAXMOVES = 150, and $size$ = 4 ; the two other instances being out of reach from complete searches. Five methods have been compared : the *Depth First Branch and Bound* (DFBB), LDS with only one iteration, two variants of LNS (LNS/CP/GR and LNS/LDS+CP) which mainly differ by their reconstruction mechanism, and VNS/LDS+CP. For the two variants of LNS, the best percentage of unassigned variables represents 60% of the total number of variables.

Once again, $discrep$ = 4 constitutes the best parameter setting for VNS/LDS+CP. The behavior of VNS/LDS+CP is quite similar to those observed on instances 6 and 7, which clearly offers better performance profiles than the other competing methods. On this instance, VNS/LDS+CP almost finds the optimum in a very short time at each run. We can draw some remarks :

- pure LDS (or only one iteration of LDS) is completely inefficient in an *anytime* context, even if LDS leads to better performance profiles than DFBB.
- the efficiency of VNS/LDS+CP is certainly partly due to the reconstruction mechanism. Indeed, the comparison between performance profiles of LNS/CP/GR and LNS/LDS+CP show that, with an effective propagation (lower bounds computation) combined with LDS, LNS obtains a gain in quality of 7% after 20 seconds.
- the use of variable size neighborhoods is a key point in the efficiency of VNS/LDS+CP. Indeed, it produces better performance profiles than LNS/LDS+CP. After 2 seconds, VNS obtains a gain in quality of 9% ; this gain reaches a value of 5% after 15 seconds. This amelioration comes from the speed of the exploration of relatively small neighborhoods (in particular during the first moves of VNS). This greatly improves the quality of the computed *upper bound* which will enable a better pruning at the next reconstruction step. This is not the case for LNS, which needs much more time to obtain good upper bounds, due to the important size of the neighborhoods.

To evaluate the improvement speed of the quality of solutions provided by VNS and LNS, figure 4 compares

VNS/LDS+CP and LNS/LDS+CP, in term of the average percentage of improvement, $Q_p(t)$, with respect to the initial cost, C_0, and in term of the number of attempted moves (in percentage). $Q_p(t) = 100 \,.\, (C_0 - C(t))/C_0$, where $C(t)$ is the valuation of the best solution found at each instant t.

As depicted in figure 4, VNS/LDS+CP provides the better improvement of the initial cost C_0. This difference in quality (in favor of VNS) is very significant during the first instants. After only 4 seconds of computation, VNS improves C_0 by 35%, with a percentage of attempted moves equal to 4% of MAXMOVES. For comparison, LNS needs 15 seconds and practically three times more of local moves ($\simeq$ 14%) to reach the same improvement of quality. Moreover, the number of moves attempted by LNS grows larger with time, while for VNS this number remains low and quasi constant.

6 Resource allocation in ATM networks

VNS/LDS+CP has been successfully applied to a real-life *anytime* resource allocation problem. This application, developed for France Telecom R&D, takes place in an ATM (Asynchronous Transfer Mode) network administration context. The problem consists in planning demands of connection over a period of one year. Reservations must be computed within *at most one minute* per demand.

A new demand is accepted if both bandwidth requirements and quality of service are satisfied. If not, we try to *reroute* some already accepted connections in order to find a route for the new demand. If rerouting fails, the demand is rejected. First, a routing heuristic computes shortest paths which minimize capacity violations on the links. From these violations, a sub-area of the network that can be modified by rerouting some already accepted connections is determined and modelled as a VCSP. Then, rerouting is performed with VNS/LDS+CP. Experimental results show that rerouting with VNS/LDS+CP enables to admit an average of 67% of demands that are rejected with a greedy algorithm without rerouting [Loudni *et al.*, 2003].

7 Conclusions and further work

VNS/LDS+CP is a new hybrid method for solving constraint optimization problems (modelled as VCSPs) in an *anytime* context. It has been successfully applied to a real-life *anytime* resource allocation problem in ATM networks (67% of rejected demands are now accepted).

Experiments on the CELAR benchmarks have shown that VNS/LDS+CP provides better performance profiles than the other competing methods. Moreover, our method is robust : for all considered instances, the best parameters setting is always the same. Finally, VNS-LDS+CP is stable : the distribution of the medians $1/4$, $1/2$ and $3/4$ shows that the variation of the evolution of solution quality is negligible within the fifty runs. However, the difference between the best run and those of the medians is relatively distant for instance 7.

From an anytime point of view, in addition to the properties of monotonicity, interruptibility and measurable quality, our performance profiles also verify the important property of *diminishing returns*, which guarantees that the improvement in solution quality is larger at the early stages of computation.

Performance profiles show a decelerating phase leading to a quasi stable plateau. To escape from this plateau, we intend to study two possibilities : to cooperate with other efficient pure local search method [Voudouris and Tsang, 1998] or to re-start our method.

In this paper we have mainly focused on the design of *anytime* algorithms. Works remain to be done concerning techniques for monitoring and control (see [Hansen and Zilberstein, 2001]). We are currently investigating such techniques for meta-level control. The use of *nogoods* would help to obtain more relevant neighborhoods.

References

[Bistarelli *et al.*, 1999] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie H. Fargier. Semiring-based csps and valued csps: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.

[Boddy and Dean, 1994] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–295, 1994.

[Cabon *et al.*, 1999] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners.W.K. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.

[Caseau *et al.*, 1999] Y. Caseau, F. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems (meta-programming for meta-heuristics). In *Proceedings of CP-99*, number 1713 in LNCS, pages 144–158, VA, USA, 1999.

[Focacci *et al.*,] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. *To apear in F. Glover and G. Kochenberger (Eds.), HandBook on Metaheuristics, Kluwer Academic Publishers.*

[Hansen and Mladenovic, 1997] P. Hansen and N. Mladenovic. Variable neighborhood search. *Computers And Operations Research*, 24:1097–1100, 1997.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.

[Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, Montreal, August 1995.

[Laburthe, 2000] François Laburthe. Choco: implementing a cp kernel. In *CP00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems (TRICS)*, Singapore, September 2000.

[Larrosa *et al.*, 1999] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for solving max-csp. *Artificial Intelligence*, 107(1):149–163, 1999.

[Lobjois *et al.*, 2000] L. Lobjois, M. Lemaître, and G. Verfaillie. Large neighbourhood search using constraint propagation and greedy reconstruction for valued csp resolution. In *Proceedings of the ECAI2000 Workshop on Modelling and Solving Problems with Constraints*, 2000.

[Loudni *et al.*, 2003] Samir Loudni, Patrice Boizumault, and Philippe David. On-line resources allocation for atm networks with rerouting. In *Proceedings of Integration of AI and OR Techniques in Constraints Programming for Combinatioral Problems (CP-AI-OR'03)*, Montréal, Canada, May 2003.

[Prestwich, 2000] S. Prestwich. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In *Proceedings of CP 2000*, number 1894 in LNCS, pages 337–352, Singapore, September 2000.

[Schiex *et al.*, 1995] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95*, Montreal, August 1995.

[Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of CP-98*, pages 417–431, 1998.

[Voudouris and Tsang, 1998] C. Voudouris and E. Tsang. Solving the radio link frequency assignment problem using guided local search. In *Proceedings of NATO symposium on Frequency Assignment, Sharing and Conservation in Systems*, Aalborg, Denmark, 1998.

[Zilberstein, 1996] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.