RAIRO Operations Research
RAIRO Oper. Res. **40** (2006) 77–95
DOI: 10.1051/ro:2006018

COMBINING CONSTRAINT PROPAGATION AND META-HEURISTICS FOR SEARCHING A MAXIMUM WEIGHT HAMILTONIAN CHAIN

Yves Caseau[1]

Abstract. This paper presents the approach that we developed to solve the ROADEF 2003 challenge problem. This work is part of a research program whose aim is to study the benefits and the computer-aided generation of hybrid solutions that mix constraint programming and meta-heuristics, such as large neighborhood search (LNS). This paper focuses on three contributions that were obtained during this project: an improved method for propagating Hamiltonian chain constraints, a fresh look at limited discrepancy search and the introduction of randomization and de-randomization within our combination algebra. This algebra is made of terms that represent optimization algorithms, following the approach of SALSA[1], which can be generated or tuned automatically using a learning meta-strategy [2]. In this paper, the hybrid combination that is investigated mixes constraint propagation, a special form of limited discrepancy search and large neighborhood search.

1. Introduction

Combining constraint programming and meta-heuristics is a powerful technique which requires, however, a large amount of experimental work for fine-tuning. Furthermore, those hybrid algorithms lack robustness from an industrial point of view since they are very sensitive to the typology of input problems. A small change in the input problem often requires a new phase of experimental tuning.

Received December 31, 2005.

[1] Bouygues e-Lab, 1 avenue Eugène Freyssinet, 78061 St-Quentin en Yvelines Cedex, France; ycs@caseau.com

This is the reason why a lot of attention has been paid in the past few years, especially by Minton in 96 [3], to the use of program synthesis and learning as an alternative to human tuning. This is an ambitious but exciting path, which has already shown interesting results in the field of vehicle routing [2]. The foundation paradigm is to represent search algorithms as hybrid combinations of techniques, mixing search patterns and generic heuristics, which may be seen as an algebraic term. The learning meta-algorithm explores the algebra to produce terms that represent algorithms that produce high-quality solutions.

ROADEF challenge problems are interesting candidates to apply such techniques, because they are large-scale complex problems, with an industrial background that gives a lot of sense to the quest for robustness and ease of development. The problem that was presented for the 2003 Challenge is a satellite scheduling problem, where the goal is to pick a subset of views that make a feasible tour for the observation satellite and have the maximal combined value. The feasibility constraints may be seen as distance constraints between views, thus the result may be seen as a Hamiltonian chain. The complexity of the cost function comes from the existence of subsets of views which have a combined higher value that the sum of the individual values. Compared to the previous 2001 problem, the target run-time is shorter (300s) and the solution space is denser (it is easier to find a first solution), which has enabled us to go further in the application of the "meta-heuristic learning" approach. It is interesting to note, however, that we used precisely the same combination of techniques (CP, LDS and LNS) for both problems.

In this article, we relate our approach to this satellite scheduling problem and focus on a few key aspects of this problem. First, because only a subset of the existing views is selected in the final solution, the problem is not a TSPTW (Traveling Salesman Problem with Time Window), which has been well studied by the constraint programming community [4, 5], but a Hamiltonian chain problem. One of our contributions is to propose a new "OrderedSubchains" redundant constraint that yields interesting propagation methods. This redundant propagation is mandatory to perform an efficient search based on a sub-chain branching schema, which is implemented using a limited discrepancy search. This search is, in turn, used within LNS. Another interesting contribution of this paper is a re-design of the combination algebra compared with previous papers [2, 6], since some meta-heuristics have been dropped but randomization and parallel composition has been introduced.

This paper is organized as follows. Section 2 describes the hybrid algorithm, from the constraint model to the LDS and LNS meta-heuristics. We introduce the concept of ordered sub-chains, which is the basis for strengthening the propagation for ordering and time-window constraints in a problem where only a subset of the initial tasks will belong to the final solution. We present a simplified version of limited discrepancy search [7] which has proven successful with a large range of problems and an implementation of LNS for this ROADEF problem that is very similar to Shaw's implementation for VRPTW[8, 9]. Section 3 focuses on the

algebra for describing hybrid algorithms and the use of step-by-step program synthesis, which is the solution that we found to solve the run-time/experience-time problem that we had reported with large-scale problems in an earlier paper. We also describe the introduction of de-randomization, which actually improved the efficiency of the random selection of the fragment that is removed and re-inserted in the LNS schema. The last section draws on the strengths and weaknesses of the approach that is presented in this paper to suggest directions for future work. We introduce a new research project, named CECILE, whose aim is to improve substantially the speed and robustness of the development of the algorithm, while minimizing the impact on the quality of the solution that is produced.

2. A constraint-based approach

2.1. Model

The ROADEF problem is described in [10] (a formal model may also be found in [9] together with a constraint propagation model which is similar to the one presented here). It is defined by a set of views that must be selected for an earth-orbiting satellite, such that the succession of shots is feasible and such that the overall value of the views that are selected is maximal. There exists a time distance between views, which reflects the travel time of the satellite from one observation position to another, and a set-up time that represents the positioning of the camera in the proper direction. The value that is associated to a set of views is complex because it takes two "aggregation" features into account. First, some views are part of a larger set of views, a group, so that the total value is a convex function of the ratio of the global view that is actually made. Second, some views are part of pairs of stereoscopic views and must be taken as pairs to be worthwhile.

The model that we use is a TSPTW model, that is, a Traveling Salesman Problem with time windows. The principle is the construction of the solution tour through a binary relation next/prev (x.next is the task that is the successor of x within the tour). The elementary branching decisions follow a "x.next := y" pattern (y is the successor of x, thus y.prev := x). Each view is seen as a "task", which is a part of an "Order" object that represents, when necessary, the group of views that constitute a global view. We use two virtual tasks TStart and Tend to represent respectively the start and the end of the tour.

For each task, we build and maintain the following attributes:

- the time window (*minStart*, *maxStart*) that represents the time constraints, both the initial window that is part of the problem and the additional new constraints inferred by propagation;
- the *next*/*prev* pair (once a choice is made) and the list of possible neighbors (a neighbor is a possible successor), which is an ordered list (from the closest to the furthest). The distance to the closest neighbor is kept and is called the diameter of the task. It plays a classical role in the computation of the lower bound;

- features that are inherited from the view: duration, existence of a stereo "twin", order to which the task belongs;
- a status (*in*, *out*, or *unknown*), which says if the task is definitely part of the solution, definitely not part of the solution, or if it is still unknown whether the task will be used or not.

In addition to the distance constraint, the ROADEF problem requires two additional types of constraint. Each view may actually be taken in two possible directions. The algorithm must pick one direction, since the direction has an impact on the set-up and on the starting/ending position of the satellite, but they are obviously exclusive: one only need one view for each image. This is modeled simply by creating a pair of tasks for each view and an exclusion constraint between these two tasks. Similarly, we model the "stereo" constraint by saying that if one view from a stereoscopic image is part of the solution, then the other view must be part of the solution. We enforce arc-consistency on this constraint, which means that if a stereo view is part of the solution, we ensure that there is a possible assignment (a position in the solution chain) of its "stereo twin" that does not cause a contradiction.

2.2. Propagation

Propagation of constraints is defined according to the following set of events:

- selecting an edge (selecting the successor y of a task x);
- deleting a neighbor (a possible successor) for a task x;
- dropping a task (changing its status to "out");
- reducing the time-window associated to a task (increase the *minStart* or decrease the *maxStart* attribute).

The propagation rules are similar to the ones developed for the TSPTW problem [4], they perform straightforward updates following four propagation paths:

1. the time-windows are reduced according to the set of possible neighbors (next/prev). The associated invariant is that the min and max values for the starting time are compatible with the selection of one neighbor;
2. the list of possible predecessors and successors is reduced when the time-windows do no longer allow for the duration of the link. It is also reduced by the propagation of the AllDifferent constraints associated with the next/prev relations;
3. the constraints that states the absence of sub-tours is enforced by removing the links that would create a loop from the possible neighbor list;
4. a simple lower bound of the total length of the final chain is maintained.

The master propagation method is the one that deals with the selection of a link (*i.e.*, making a decision of the kind $x.\text{next} := y$). This method may be summarized as follows: the time bounds are modified by the distance between x and y, the ancestor of x in the solution (sub-chain) is removed as a neighbor of y to avoid the creation of cycles. There is, however, a major difference with the time-window propagation since we can only apply the propagation rules if we

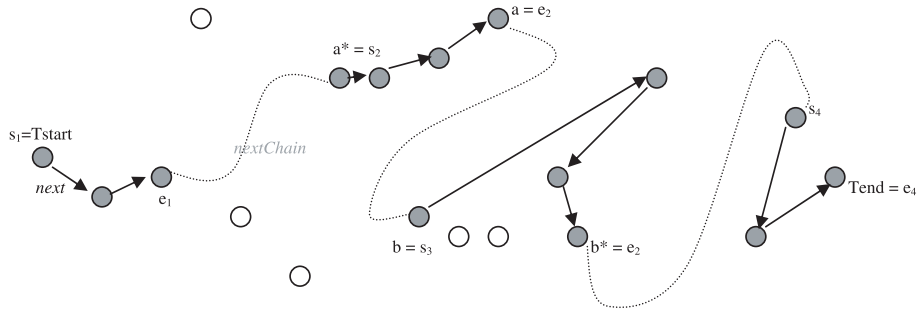

FIGURE 1. Sub-chain ordering.

know that a task is definitely part of the solution. If we want to use the same branching scheme that was described in [4] for TSPTW, the weakening of the propagation rules is such that dead-ends occur in the search tree that requires expensive exploration. It turns out that the propagation rules that are strong enough to guarantee the convergence towards a feasible solution in the TSPTW case are too soft in the Hamiltonian chain case. A simple solution is to revert to a "chronological" branching scheme: start with the first task (TStart) and build the final chain by choosing the successor nodes from start to end. This is in contrast with the approach of [4] where we select the "most critical" task x with a combination of number of neighbors and "regret" (difference in distance between the closest neighbor and the next) and we branch on "x.next $:= y$" or "x.next $\neq y$". This binary branching schema is much more powerful than chronological branching but it yields to the creation of multiple sub-chains, which, in the Hamiltonian chain, may become tricky to join together.

We solved this problem with the introduction of a redundant constraint, which we call SCO for sub-chain ordering in the remainder of the paper. A sub-chain is a set of tasks that have been selected in the solution (their status is "in") and that linked with the relation *next*. The sub-chain C_i start with the task s_i and ends with the task e_i, as shown in Figure 1. The ordering that we use here is the time ordering induced by the time windows: we say that $x \leq y$ if $x.minStart +$ distance$(x, y) \leq y.maxStart$.

It is important to notice that our contribution is not the principle of "subchain ordering" (composing a tour from the proper ordering of various subchains), which is used in many local optimization methods, such as the Tabu heuristic of [11], but the propagation method associated to this global constraint (SCO).

The SCO constraint is the conjunction of two properties:

(P1) All sub-chains must be ordered: For each pair of sub-chains C_i and C_j, either $e_i \leq s_j$ or $e_j \leq s_i$.
The total order among sub-chains is represented with the *nextChain*/ *prevChain* relation: if C_i is just after C_j then $e_j.nextChain = s_i$.

(P2) Each well-ordered task must have a closest neighbor which is compatible with the adjacent sub-chains. A task that is not part of a solution but that may only fit between two sub-chains C_i and C_j is said to be well-ordered. The closest neighbor y of such a task x is compatible with C_i and C_j iff either:
- $y = s_j$ and $d(e_i, x, y) <= y.maxStart$;
- $d(e_i, x, y, s_j) <= s_j.maxStart$

where $d(x_1, ..., x_n)$ is the earliest arrival time at x_n if we start at x_1 at the earliest time and take distance and time window constraints into account. This simply says that there exists a feasible path from e_i to s_j that goes through x and y.

Enforcing the SCO property reduces the search space and does not remove any solution since any Hamiltonian chain satisfies the SCO constraint by definition (one single chain). Enforcing this constraint requires two steps:

(1) *SCO propagation.* The propagation step uses a method *SCOenforce*(...) whose goal is to detect well-ordered tasks, maintain the property P2 and remove impossible links (*deleteNeighbor*) or tasks (*dropTask*).
(2) *SCO branching selection.* There is a restriction on the "smart" branching task selection heuristic to ensure that (P1) is satisfied.

The branching selection can be stated simply as follows: we select a pair (x, y) to branch on the decision (x.next := y or x.next $\neq y$) only if either:

- x or y are already part of the solution (their status is "in");
- x is well-ordered.

The propagation for SCO consists of extending the *link* (x, y) propagation method with another propagation method that is called *SCOenforce*. When we create a new link in the solution, we first perform the usual TSPTW propagation as explained earlier, then we determine easily if this new link creates a new sub-chain (when x is well-ordered), extends an existing one or merges two sub-chains. We then consider the beginning s_i of the resulting chain C_i to which x belongs, and the end of this chain e_i. We call *SCOenforce* (s_i.prevChain,s_i) and *SCOenforce* (e_i,e_i.nextChain) to examine the tasks that could fit between respectively the chain that is "before" C_i and C_i, then the tasks that could fit between C_i and the chain that is "after" C_i.

More precisely, *SCOenforce* (a, b) examines all tasks that are not already part of the solution or already dropped and looks if they could fit before the chain that ends with a, after the chain that starts with b or in the middle. If a task x can only fit between a and b it is well-ordered and the filtering (P2) may be applied. Otherwise, we apply the exclusion rules: if a task may not be before the start of a sub-chain, it must be after its end.

TABLE 1. Impact of Hamiltonian propagation.

Problem (*cf.* 3.3)	Ch + simple p.	Ch + SCO	B&B + simple p.	B&B + SCO
4	417	486	401	496
5	449	439	386	523
6	405	405	_	498
7	282	337	_	610
9	160	175	89	159
10	_ (no sol)	284	_	352

Here is a pseudo-code description of the *SCOenforce(x, y)* method:

```
// examine tasks that could be well-ordered between a and b
SCOenforce(a,b)
Let a* = start of the chain that ends with a, b* = end of the
chain that starts with b in
for all tasks t with status ‘‘unknown’’
if (t ≤ a*) (if not(a ≤ t) pushBefore(t,a*) else mark t
as not(well-ordered))
else if (b* ≤ t) (if not(t ≤ b) pushAfter(t,b*) else mark t
as not(well-ordered))
else (if (a ≤ t and t ≤ b)
    (mark t as well-ordered,
    enforce the interval: pushAfter(t,a) and pushBefore(t,b),
    filter all neighbors of t to make sure that they will
    satisfy (P2) if they become closest neighbors)
  Else dropTask(t))
```

Figure 1 shows the various tasks that are used in the previous algorithm.

The existence of the SC ordering has an additional (trivial) impact on the time-window propagation since lower/upper bound updates may now be propagated both on the *prev*/*next* links (the sub-chains) and the *prevChain*/*nextChain* links (the ordering).

The other propagation method that is impacted by the SCO constraint is the neighbor deletion method since whenever the closest neighbor is removed for a node that is well-ordered, the new closest neighbor must be checked to satisfy the consistency principle (P2).

Table 1 shows the improvements that are obtained with SCO with either chronological or “smart” branching schemas. Another way to measure the improvement

is the fact that SCO ensures that the branch-and-bound algorithm finds a solution for each of the 20 ROADEF problems without any backtrack, which makes it a powerful greedy heuristic.

It is also interesting to notice that our experiments with "shaving", a domain consistency technique that was shown to be successful for job-shop scheduling by Martin [12] and that we found very successful in the previous ROADEF challenge, did not bring any improvement for this problem. Shaving may be described as removing extreme values from a variable domain by propagating the choice of this value and removing it whenever the propagation yields a contradiction. The absence of benefits from this technique (which main drawback is to be computationally expensive) confirms that this ROADEF problem is much closer to a TSP than to a scheduling problem.

2.3. Search with LDS

The search for a solution is made using an incomplete branch-and-bound scheme, very similar to the limited discrepancy search of [7], although we have found that the simpler approach that is presented here is more efficient and more generic than the original design of Harvey and Ginsberg. This algorithm has been used for large-scale routing problems since 1993 and has shown very competitive results (for instance, [2]).

The LDS principle is to develop a bare search tree with few nodes and branches, by restricting the branching points to a few decisions, according to the depth in the search tree (with the implicit assumption that the most important choices are made earlier) and according to the heuristic function that selects greedily the tasks that need to be inserted (with the implicit assumption that the heuristic is most often right). More precisely, our generic LDS scheme is a method to transform a greedy heuristic defined by a choice heuristic f into a limited search algorithm that shows a performance improvement, when we increase the size of the search space, that is far superior to a classical branch-and-bound. The efficiency of this scheme is tightly linked to the efficiency of the greedy heuristic: applied to a poor heuristic, it gives poor results; applied to a good heuristic, it provides interesting results, especially for large scale problems (such as vehicle routing with 1000s of nodes). Here we suppose that the heuristic is defined by a valuation function f which tells which task should be inserted into the solution, and that the insertion means to link the task to its closest successor. However, the scheme can be extended easily to a non-binary branching schema.

The LDS extension of the greedy heuristic defined by f may be described as follows. It is characterized by two parameters, k and d, which represent respectively the maximal number of branching to build a solution (hence, with a binary branching schema, a bound of 2^k for the size of the search tree) and the "discrepancy" which is the difference in the heuristic value that will cause a branching (measured as a percentage).

```
LDS(k,d)
    Pick x such that the SCO branching constraint is satisfied
    and f(x) is maximized
        Let y := closest_neighbor(x)
             Create a choice point // not for branching
             (cf. explanation)
             Assert (x.next := y) // and propagate
             If no contradiction has been detected, recursive
             call of LDS(k,d)
             Return to choice point (backtrack)
             If k > 0 and if there exists x2 such that
             f(x2) > f(x) * d, // d%
                 Create a choice point // optional branching
                 Assert(x.next != y)
                 If no contradiction has been detected, call
                 LDS(k - 1, d)
                 backtrack
```

There is one subtlety with the use of "choice points". A choice point is a virtual store of the current state that supports further restore (backtracking). This mechanism is supported by most constraint programming languages. Here, we use choice points for two different purposes. The first case is a contradiction handling pattern: propagation may raise "contradiction exceptions" that yield a backtrack. The second case is the conditional branching that correspond to the exploration of the reduced search tree.

In this pseudo-code description, *assert(decision)* means to generate the event that correspond to the *decision*, and then call the propagation methods recursively on this event and all the events that are generated by propagation, until no new further deduction can be made. Then a call to the upper bound method is made, which raises a contradiction if the upper bound is lower than the best value found so far (as in any branch-and-bound algorithm).

Computing an upper bound for a partial solution of this problem is difficult because of the complexity of the objective function. The principle that we use here is to use a relaxation of the problem into a scheduling problem, for which a bound is obtained with a greedy algorithm. We associate to each task in the original problem a scheduling task whose duration is the travel time between the original task and its closest neighbor. We then use three ideas:

(1) Use a simplified "task interval" analysis [13] to find which tasks, for each order, are the most profitable and may participate to the final solution.
(2) Use the convex nature of the value function associated with each order to determine an upper bound of the value per second for each of these tasks
(3) Each task is then virtually decomposed into time units with an associated value and the solution is composed in a greedy manner by picking for each time interval the units that bring the most value. Alternatively, we could solve the continuous relaxation of the associated knapsack problem, but

TABLE 2. Comparing LDS with a regular search.

Problem	Search@ 5 K	Search@ 20 K	Search@ 100 K	LDS (7)@ 5 K	LDS (7)@ 20 K	LDS (7)@ 100 K
4	480	480	480	490	496	496
5	511	513	515	517	523	540
6	428	428	428	610	610	613
7	415	415	425	471	498	498
9	155	155	155	157	159	162
10	313	323	329	328	352	352

we found that the pre-emptive relaxation (seeing each task of duration n as n units) is simpler to work out with our existing data structures.

This bound computation is fairly complex, with a first step that combines the two first ideas to associate to each task a "value per second" and a second task which fills the time schedule with pieces of the task. The greedy process described in (3) is implemented with a loop that successively merges tasks that have intersecting time intervals until one single combination is obtained.

Table 1 compares the two branching methods (Ch for chronological and B&B for the "smart branch&bound" with a dynamic insertion ordering) and the effect of SCO (simple p. stands for the simple TSPTW rules of [4] and SCO for the addition of Sub-Chain Ordering). The problems are the 6 difficult problems of the first set of instances of the ROADEF challenge (the correspondence with the full instance names is given in Sect. 3.3), the value in the table is the objective function divided by 1 000 000). The effectiveness of SCO is obvious, but is clearly more spectacular when combined with a dynamic branching scheme (the effect of the improved propagation is limited with a chronological approach).

The previous results were obtained with a LDS (7, 80) with a cut-off at 20 K backtracks. Table 2 shows the effectiveness of the LDS scheme. We compare a regular branch-and-bound scheme (search) with LDS at different level of search tree (there is an automatic cut-off after 5 K, 20 K and 100 K backtracks). We see that LDS is both much more efficient for a given size of search tree but also makes a better usage of additional search time.

2.4. LNS

Large Neighborhood Search (LNS), also called "shuffle" [14] in the context of jobshop scheduling, is an optimization meta-heuristic that is based on using LDS to reconstruct a better new solution from an old one, thus defining a structure of large neighborhoods (the distance between such two solutions may be important). More precisely, this technique consists of removing a fragment of the initial solution, then re-introducing it using a LDS search for an improvement. The key part of this strategy is to define which fragment should be removed (or which part should be kept).

For this ROADEF 2003 problem, we have used a simple approach, which is based on removing a fragment that is defined by a time interval. We pick a small sub-chain (size 1 or 2), then add all initial neighbors and tasks that are linked with a "stereo" constraint. The choice of neighbors may be randomized slightly, following a suggestion of Shaw in [8]. The fragment is then reduced to match a given size. This size is the key parameter for LNS: for instance, in [8] the best strategy is to try all possible sizes in an incremental manner. For the ROADEF 2001 problem, we enriched this strategy with a rotation: increase from 5% of the solution up to 90% (!) and then revert to 5% and so on. For this problem, which has a much simpler structure than the 2001 problem, using fragments from 5% to 25% of the solution seemed to provide the best results.

The LNS algorithm may be described as follows:

```
LNS(k,n)
Repeat k times
  pick a set S of tasks of size n,
  re-initialize the tasks from S,
  find a solution with LDS(_,_)
  if we find a solution which is better than the current
  solution, this solution becomes the current solution
```

The re-initialization step consists of removing a task from the current solution if it is part of it, or resetting the task's status to "unknown" if was dropped out, and then re-computes the initial time window and the original set of neighbors. One may notice that there is a proper tuning needed with the LDS method to find which parameters work best for the LNS. This will be addressed in the next section. The efficiency of the LNS method is directly bound to the strength of the constraint propagation, as noticed already by Applegate and Cook in [14].

2.5. Control

The overall strategy used in this paper is to produce a first solution with a LDS algorithm and then optimize it through successive application of the LNS algorithm. Experience shows that, although it is important to produce a strong propagation strategy and to tune the LDS parameters to make the LNS efficient, it is not mandatory to get a very good first solution. Indeed, the "repair" capacity of the LNS algorithm is such that it is faster to improve a mediocre solution up to a "good" one than to tune the LDS so that it produces a "good" first solution. To ensure an "anytime" behavior, we encapsulate the LNS loop within another loop, which is controlled by the run time and which stops once the target time for the challenge has elapsed.

It is interesting to notice that it seems that the quality of the bound, which is crucial to produce a first solution of high quality with LDS, is less important for "repairing" an existing solution within the LNS framework. This is an insight that will influence future experiments with the CECILE framework (*cf.* Sect. 4) since a large part of the development time both for the ROADEF 2001 and ROADEF 2003 challenges was spent on the implementation of an upper bound method.

3. Algebraic representation of hybrid algorithms

3.1. Combination algebra

The first step, to perform meta-heuristic combination and synthesis, is to use the algebraic modeling of hybrid algorithms that was proposed in [1]. We use a term algebra that is generated by the following operators [3]:

- **LDS (f,h,N,d)** applies the LDS algorithm to build a solution, using a heuristic choice function represented by the integer f, an explicit control of the size of the search tree that is represented by h, a maximal number of branching for a given solution (the k parameter from Sect. 2.3) and a parameter for the LDS branching condition d.
- **FORALL (f,h,N,d,O)** is a variant from the previous LDS algorithm, where the optimization algorithm represented by the sub-term O is applied for each solution that is produced by the LDS algorithm. Note that capital letters are used here to represent sub-terms (*i.e.*, N is a term that denotes an integer).
- **LNS (h,N,d,L)** applies one iteration of the LNS algorithm that was presented in the Section 2.4. The integer h represents the heuristic that must be used to select the removed fragment (hence we can experiment with various heuristics and compare them). N represents the desired size for the removed fragment, d is the level of randomization and L is a LDS sub-term which is used to re-build a new solution once the fragment is removed.
- **DO (B, O)** is used to compose a term B which builds a first solution (a LDS or a FORALL) and a term O that optimizes a current solution.
- **THEN (O1,O2)** is a sequential composition of two optimization sub-terms O1 and O2.
- **LOOP (N,O1)** applies N times the optimization algorithm represented by O1.
- **UNTIL (t,O1)** applies O1 until t seconds of run-time have elapsed.
- **BEST (O1,O2)** is a parallel composition of two optimization algorithms. Each algorithm is applied to the current solution and the best of the two new solutions is kept.
- **RANDOM (n1,n2)** represents the random selection of an integer between n1 and n2.

For instance, the term that our tuning experiments finally produced and that we used as our submission to the ROADEF challenge is:

```
DO(LDS(1,2,7,80)
    THEN( UNTIL(180,LOOP(100, LNS(2,RANDOM(20,40),90,
    LDS(4,1,7,80)))
        UNTIL(100,THEN ( LOOP(100, LNS(2,RANDOM(20,80),80,
        LDS(1,2,7,80)))
            LOOP(100, LNS(4,RANDOM(80,160),80,LDS(4,0,7,80)))
```

TABLE 3. Results with the first set of instances.

	File	*value*	*Best (*)*
1	instance_2_9_36	10423440	10423440
2	instance_2_9_66	115710959	*115710959*
3	instance_2_9_170	191358231	*191358231*
4	instance_2_13_111	558385351	*563597071*
5	instance_2_15_170	708993780	*717239040*
6	instance_2_26_96	970771000	*1005301900*
7	instance_2_27_22	940337410	*966056500*
8	instance_3_8_155	121680360	*121680360*
9	instance_4_17_186	181302215	*185406780*
10	instance_3_25_22	425971220	*425983220*

(*) The best solutions found during the first step of the challenge are reported in the last column [10].

This term represents the following strategy:

(1) Build a solution with LDS (1,2,7,80). The integer 1 means that we use the first heuristic function for selecting the task that must be inserted (among a list of 4). The integer 2 sets up the maximal number of backtracks using an exponential increment (the higher this number, the larger the search tree).
(2) Apply for 180s an optimization algorithm defined by the iteration of a LNS step that removes a fragment of size between 20 and 40 and uses LDS (4, 1, C (7), 80) to rebuild it (note the change of heuristic).
(3) Apply for 100s a sequential combination of two similar LNS terms, with different fragment sizes (20 to 80 and 80 to 160) and different LDS tuning (note that we use a "lighter" LDS for larger fragments).

This term is definitely more complex than the first term that we submitted for the first phase of the challenge, which was tuned step-by-step in a classical manner. It also produced an overall improvement of more than 2 % on the average solution value, which is significant (on the first set of instances – *cf.* Table 3 – the difference between our approach and the best known solution is less than 1%). This complex term was generated using a learning process that is not fully automatic yet but could be, using the techniques that are presented in [2]. We wrote scripts to simulate each step of the learning "meta-algorithm" and performed the selection of the best generated term manually.

The main contribution of this experiment to our long-term goal of automatic synthesis and tuning of hybrid algorithm is that we found a method for solving the run-time barrier that we had identified in a previous paper [2]. We select a pattern such as:

DO(t1, THEN(UNTIL(a,t2), UNTIL(b,t3))),

and we tune the various sub-terms t1, t2 and t3 separately using shorter run-times. If we recall that the time that is necessary to produce an algorithm with run-time X varies from 1000X (tuning) to 10 000X (generation), with our current learning strategy, one may see that this decomposition approach is necessary to get a feasible result. This requires the addition of a context to our initial design, since, for instance, the term t3 must be found, once t2 is stable, as a post-optimization of the solution produced by t2.

3.2. Randomization and de-randomization

It is striking that we use randomization in many places for these hybrid combinations. Because the run-time is relatively short compared with the size of the problems, we tried to replace the random generators with pseudo-random (deterministic) generators that provide a better sampling (more uniform and more systematic when called only a few times). Indeed, for selecting a fragment for LNS, "true randomness" has no benefits: we use random selection as a cheap implementation of a complex iteration loop that would generate all possible subsets, while avoiding the "accumulation point" effect. We replaced the calls to the system random functions with our own pseudo-random generators, while using one generator for each separate usage in the algorithm. We have implemented a very simple pseudo-random generator pattern, using a counter with variable increment and a modulo operation.

The result is very positive: the pseudo-random generators provide better quality results than the "true random" ones, when the run-times are kept short. Obviously, as one might expect, the difference reduces as the run-time grows. However, in the context of the ROADEF challenge, we found pseudo-randomization to offer a dual benefit: it makes the hybrid algorithm a deterministic algorithm, which is easier to evaluate (and much easier to manage with a learning approach [2]), and it produces better quality results.

3.3. Results

Table 3 gives the results that we obtained with the algorithm described in Section 3.1, using a Pentium III at 800 MHz. Problems 1, 2, 3 and 8 are solved optimally by the branch-and-bound step. The target run-time is approximately 300s, and the quality of the results is sensitive to the speed of the machine (a machine with a faster memory cache produce betters results !). These results are within 1% of the best algorithms in the challenge.

If the algorithm is decomposed as DO(T1,THEN(UNTIL(180,T2), UNTIL(200, THEN(T3,T4))), it is interesting to compare the values obtained at different stages of the computation: T1 (after 20 s), T1 + T2 (after 200 s). Table 4 also shows the values that are obtained with a single LNS strategy: respectively T1+T2, T1+T3 and T1+T4 for 300 s.

These results show the benefits obtained with our "learning strategy", since the hybrid term which is produced outperforms what one could have obtained through

TABLE 4. Partial Results (intermediate and alternate).

	T1 (1 to 19s)	T1+T2	Full (300s)	T1+T2(300s)	T1+T3(300s)	T1+T4(300s)
1	10423440	10423440	10423440	10423440	10423440	10423440
2	115710959	115710959	115710959	115710959	115710959	115710959
3	191358231	191358231	191358231	191358231	191358231	191358231
4	480457811	555772970	558385351	555772970	547961000	532722200
5	508174928	708993780	708993780	708993780	700678000	664578000
6	428080475	970771000	970771000	970771000	935582000	930698000
7	420258660	940337410	947340350	940337410	959404000	855073000
8	116531810	121680360	121680360	121680360	121680360	121680360
9	155709805	181302215	181302215	181302215	181302215	184401220
10	323038406	425971220	425971220	425971220	425971220	420510000

TABLE 5. Results with another set of instances (base X).

Instance	Solution	Gap(Sol-Ref)/Ref(%)	Solution TNO-PEL
2_28_111	854994860	33.18	872246792
2_28_140	828074890	20.19	833061980
2_28_155	932454050	23.53	952267030
2_28_170	925772900	43.82	952144183
2_28_37	945617600	49.41	987649761
2_28_66	918358259	33.75	945737319
2_28_7	965783840	35.09	977811340
2_28_81	859021950	17.97	878847950
3_28_155	450543180	18.41	457063785
3_28_96	458106682	15.48	458107362
Synthesis/total	8138728211	**29.08**	8314937502

a manual tuning strategy. We get similar findings to what was observed in [6]. We also see that the algorithm converges pretty rapidly since the values obtained after 200 s are almost as good as those obtained after 300 s. If the running time is increased to 600s, the only improvement we get is with the 9th problem for which we get a solution 185 406 780 (same as the best known), but no other improvement is found, even if the time limit is increased to 1200 s. We will discuss this absence of improvement at the end of this section.

Table 5 shows the results that were obtained on the "hidden" set of instance problems that was used by the challenge jury (from the result web site [10]). This table shows a similar relative positioning (approx 1 to 2% from the best algorithms in the challenge) and a true improvement (29%) compared to the reference solution that was provided. To our knowledge, ours is the only approach that is heavily based on constraint programming, whereas most other approaches were based on Tabu algorithm or simulated annealing. We believe that the difficulties found by other constraint-based approaches, including an alternate method developed in our research lab, are due to the relevance of SCO for Hamiltonian chain problems.

The overall evaluation of our approach as a contribution to solve the ROADEF 2003 problem is, therefore, mixed. On the negative side, it is one of the worst finalist solutions, where the first three use local search approaches. We have included in the previous table the values obtained by the challenge winner (TNO-PEL, Kuipers [10]). On the positive side, the gap is small in comparison with the differences that were observed earlier in the challenge.

Interestingly, we may now compare our approach with the state-of-the-art of optimization heuristics for 4 problems: the vehicle routing problem with time windows, which is a classical problem for which there is a rich literature, the frequency allocation problem that was proposed as the 2001 ROADEF challenge, this 2003 problem and the car sequencing problem which was proposed for the 2005 challenge. There are three distinct situations:

- The 2001 problem was a hard problem, from a satisfiability point of view. The use of complex constraint propagation was a significant advantage and the solution that we developed using the same approach that we described in this paper did well from a competitive point of view (second in the finalist stage). The constraint propagation algorithm was able to produce interesting lower bounds.
- The 2003 problem and the VRPTW are similar: we obtain results that are close to the state-of-the-art but the best solutions are found with a tabu approach. The original paper about our algebraic/learning approach [6] demonstrates results that are among the best and obtained with a short computation time, but they are not as good as those obtained in [11].
- The 2005 problem is well suited to local optimization and the simple use of two-opt move produces high quality results. Our approach did not do well from a competitive standpoint since we did not qualify for the final stage. It is interesting to notice, though, that the results that we obtained were still much better than those obtained with the reference heuristic and within a similar distance to the best ones (1 to 2%).

The set of local optimization moves that is explored with our LNS strategy is always a superset of the moves that are implemented by the winning algorithms, but the method is much slower than ad-hoc implementation. In the 2005 problem, the speed in local move execution was found to be the most important performance factor. However, this is not enough top explain the difference in performance, since we do not get the same high quality solutions if we let the algorithm run for a day. The obvious explanation is that the monotonic decreasing "hill-climbing" meta-strategy that is used for LNS get stuck in local minimum. Even though some "large neighborhood" moves are "wide enough" to escape them in early parts of the search, the conclusion is that our algebra lacks a combinatory that would implement a non-monotonic strategy. This is a step that we are currently investigating on the 2005 problem set, and which should also prove useful for the 2003 problem.

4. Future directions

This work was implemented using the CLAIRE language [15], which is a programming language that was developed especially for developing hybrid algorithms to solve combinatorial optimization problems. CLAIRE is a high-level language, which provides complex data structures such as sets in a declarative manner. Besides, CLAIRE offers control structures that are designed to perform tree search, making the implementation from the pseudo-code contained in this paper straightforward. CLAIRE is implemented itself as a C++ compiler which generates optimized and efficient code, but also includes an interpreter which plays an important role to make the tuning of the algorithm easier.

For this challenge, we also used a library of meta-heuristics that was developed for our leaning experiments [2]. This library has already been used for various problems, ranging from vehicle routing to constrained assignments. This library contains generic algorithms for LDS or LNS, which themselves rely on insertion and deletion methods that are problem-dependent. At this stage of development, it is still a source-code library, which is used with a cut-and-paste approach. Therefore, from a research perspective, the approach presented here is average from a solution quality point of view (9th among the finalists) but interesting from the point of view of simplicity and genericity (1000 lines of code only, most of which is re-usable code, written by one person in a short amount of time – 3 weeks).

However, if we step back and look at the algorithm that we developed for the ROADEF challenge from an industrial point of view, the conclusion is the opposite. The performance of the algorithm is excellent, since it is close to the state-of-the-art, and the development process is terrible, since:

- there are too many ad-hoc "smart" propagation techniques (which an average programmer would not know);
- there is too much "know-how" needed to make the LNS/LDS combination work;
- the tuning takes too much time (much longer than the development of the algorithm).

Our next step is to try to address these criticisms more efficiently with a research project called CECILE, which is a framework for solving large-scale complex optimization problems, such as the ROADEF challenges, with a generic tool where the problem-dependent part is reduced to the minimum. Figure 2 shows an ideal vision of what CECILE could be like, where the light-colored boxes are the only problem-dependant components. The main idea for CECILE is the principle that objects should be used as foundation paradigm for integration, not constraints or logic variables. This is a departure from the "global constraint" and the "solver libraries" approach. The second idea is that there exists a catalog of branching schemas and associated choice heuristic functions that could be used with a small amount of parameterization instead of re-inventing new heuristics for each problem.

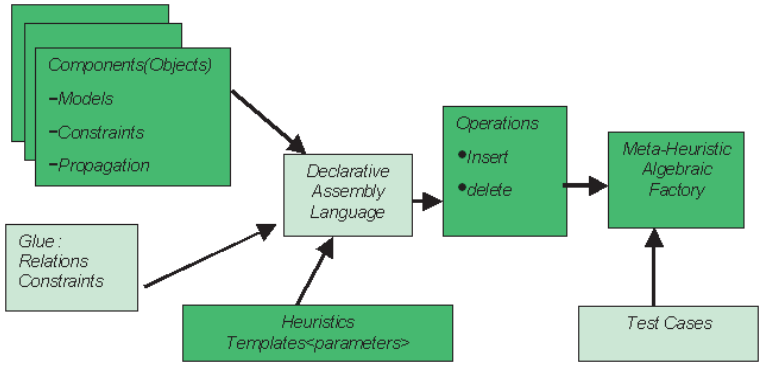

FIGURE 2. CECILE, a generic framework for hybrid problem solving.

CECILE is a long-term project. The first iteration will not include a "declarative assembly language" but is rather a framework made of generic source-code libraries. The main difference with what is presented in this paper is the library of component objects that come with their propagation and search methods. We are now re-implementing an algorithm similar to the one described in this paper with this framework. The goal is to trade-off a very small degradation in the solution quality for a very significant improvement in the development time (including tuning).

5. Conclusion

This paper presented a hybrid approach for solving the ROADEF 2003 challenge problem, using a mix of constraint programming and meta heuristics. We believe that this work contains an interesting contribution for the constraint programming community with its "Sub-Chain Ordering" (SCO) redundant constraint propagation. This method changed dramatically the results of our approach, and our algorithm would never have made it to the final stage of the challenge using a more classical constraint propagation pattern.

We also presented an interesting variation on the "limited discrepancy search" theme and showed its efficiency for the challenge problems. Last we presented a revised version of our hybrid algorithm algebra, which is both simpler and more complex, since we introduce pseudo-randomization and parallel composition. Parallel composition was not used for this problem but is notoriously useful for problems such as MAXSAT. On the other hand, we showed that de-randomization is a simple but effective improvement for our overall design. We also showed how the idea of a hybrid pattern may be used to generate hybrid algorithms for large-scale problems.

References

[1] F. Laburthe and Y. Caseau, SALSA: A Language for Search Algorithms, in *Proc. of CP'98*, edited by M. Maher, J.-F. Puget, Springer. *Lect. Notes Comput. Sci.* **1520** (1998) 310–324.

[2] Y. Caseau, G. Silverstein and F. Laburthe, Learning Hybrid Algorithms for Vehicle Routing Problems. *TPLP* **1** (2001) 779–806.

[3] S. Minton, *Configurable Solvers: Tailoring General Methods to Specific Applications*, in *Proc. of CP'97*, edited by G. Smolka, Springer. *Lect. Notes Comput. Sci.* **1330** (1997) 372–374.

[4] Y. Caseau and F. Laburthe, Solving small TSPs with Constraints, in *Proc. of the 14th International Conference on Logic Programming*. The MIT Press (1997).

[5] F. Focacci, M. Milano and A. Lodi, *Solving TSP with Time Windows with Constraints*. ICLP 515–529W (1999).

[6] Y. Caseau, G. Silverstein and F. Laburthe, A Meta-Heuristic Factory for Vehicle Routing Problems, in *Proc. of CP'99*, *Lect. Notes Comput. Sci.* **1713** (1999).

[7] Harvey and M. Ginsberg, Limited Discrepancy Search, in *Proc. of the 14th IJCAI*. Morgan Kaufmann (1995) 607–615.

[8] P. Shaw, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, in *Proc. of CP'98*, *Lect. Notes Comput. Sci.* **1520** (1998).

[9] T. Benoist and B. Rottembourg, *Upper Bounds of the Maximal Revenue of an Earth Observation Satellite*, in 4OR: *Quart. J. Belgian, French and Italian Oper. Res. Soc.*, Vol. 2, Issue 3, Oct. 2004.

[10] 2003 ROADEF Challenge: `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2003`.

[11] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J.-Y. Potvin, *A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows*, Transportation Science **31** (1997).

[12] D. Martin and P. Shmoys, A time-based approach to the Jobshop problem, in *Proc. of IPCO'5*, edited by M. Queyranne, *Lect. Comput. Notes Sci.* **1084** (1996).

[13] Y. Caseau and F. Laburthe, Improving Branch and Bound for Jobshop Scheduling with Constraint Propagation. *Combin. Comput. Sci.* **1995** (1995) 129–149.

[14] D. Applegate and B. Cook. A Computational Study of the Job Shop Scheduling Problem. *Oper. Res. Soci. Amer.* **3** (1991).

[15] Y. Caseau, F.X. Josset and F. Laburthe, CLAIRE: Combining sets, search and rules to better express algorithms. *TPLP* **2** (2002) 769–805.