

Combining constraint programming and local search to design new powerful heuristics

Narendra Jussien*

Olivier Lhomme†

*École des Mines de Nantes
BP 20722, F-44307 Nantes Cedex 3, France
Narendra.Jussien@emn.fr

†ILOG
1681 route des Dolines, F-06560 Valbonne, France
olhomme@ilog.fr

1 Introduction

In recent years many new search algorithms for solving *Constraint Satisfaction Problems* (CSP) have been proposed, ranging from improvements of chronological backtracking (the basis of many solving techniques), local search algorithms, genetic algorithms to hybrid approaches combining several search methods. Some questions come to the mind: Why so much algorithms are needed? What are their similarities? Their differences? Does a unifying framework exist? In a new search method, is everything new? To compare the different algorithms, or to design a new search method, we need to know some principles, some properties of the different algorithms. Such properties seem to be easy to study: in AI, a general search framework has been used for years. Nevertheless, if every search algorithm is an instance of this general framework, why so much search algorithms exist?

Backjumping, learning, filtering techniques, heuristics for *variable or values orderings*, repair methods, greedy methods, *Tabu Search* (ts), *gsat*, conflict-based search, revision, symmetry breaking, *Dynamic Backtracking* (dbt) [2], *mac-dbt* [12], *Dynamic Domain Splitting* (dynds) [13], *lds* [8], etc. are ideas that are quite interesting. However they are not easy to compare nor to combine; some seem to be incompatible, but how can we prove that? And the usual general search framework does not help much: based over state space and operators, it is too general to catch the precise properties we are looking for.

In this paper, we introduce the PLM algorithm which decomposes search into three components: a **P**ropagation component, a **L**earning component and a **M**oving component. We show that this generic algorithm is a useful basis for providing new search algorithms that combine constraint programming and local search: the **decision-repair** family.

Kyoto, Japan, August 25–28, 2003

2 The components of search

We identified three different components¹ that can be used to describe the behavior of many existing search algorithms:

- **P** is a **propagation** component that is used to propagate information throughout the constraint network when a decision is made during search. Two operators are needed: **filtering** *i.e.* removing parts of the search space that do not contain any feasible solution to the considered problem and **checking** if a solution can exist. **check** can answer: (a) a solution has been found, (b) no solution exists, (c) there is not enough information to conclude about the existence of a solution in the considered subpart of the search space.
- **L** is a **learning** component that is used to make sure that the search mechanism will avoid (as much as possible) to get back to states that have been explored and proved to be solution-less. Using a rough analogy with the brain, we will use two operators: a **recording** operator that learns new pieces of information and a **forgetting** operator that will make room for new information to be learnt.
- **M** is a **moving** component whose aim is, unlike the other two components, to explore the search space instead of pruning it. There are two moving operators: **repair** to be used when the current constraints system is contradictory and need some modification and **extend** to potentially add new information when no contradiction has not yet been detected but when no solution has been found.

3 A generic search algorithms

The **P**, **L**, and **M** components can be used to design a generic CSP solving algorithm (the PLM algorithm) that encompasses complete and incomplete searches, future-oriented (filtering techniques, etc.) and past-oriented (backjumping, etc.) algorithms.

Figure 1 presents our generic search algorithm based on our three components. It solves a CSP defined by a set V of variables and a set C of constraints upon the variables.

- search starts from an initial set D of decision constraints (variable assignments, precedence constraints, etc.) that may range from the empty set (typically for backtrack-based search) or a total assignment (typically for local search algorithms);
- decisions are made (using **extend**) and propagated (using **filter**) until a contradiction occurs;
- when a contradiction does occur (case **no_solution**), the information related to the dead-end (*e.g.* a conflict explanation) is learnt (using **record**), the current state is repaired (using **repair**) and some information is *forgotten* (using **forget**);

¹A formal description about these components can be found in [15].

```

procedure PLM(V, C, D)
begin
  P := {V,C,D}
  repeat
    P := filter(P)
    switch (check(P))
      case no_solution      : P := forget(repair(record(P)))
      case solution_found   : return P
      case not_enough_info  : P := extend(P)
    endswitch
  until conditions_of_termination
end

```

Figure 1: A generic algorithm for the PLM components

- search terminates as soon as a solution is found (case `solution_found`) or the conditions of termination are fulfilled. Conditions of termination can be for example a maximum number of iterations, the exhibition of a proof that no solution exists, etc.

Many well known algorithms can be seen as specializations of the PLM generic algorithm.

As a first example consider Standard Backtracking (`bt`). It does not perform any propagation and only tests the satisfiability of fully instantiated constraints. Moving upon a success of the propagation amounts to the addition of an instantiation constraint (thus making a decision). Moving upon a dead-end consists in reconsidering the latest decision. The **L** component is anecdotic since learnt information is used only once. A heuristic variable ordering is easily incorporated in `bt` by specifying the decision constraint c provided by `extend`. Interleaving arc-consistency maintenance within `bt`, to get the `mac` algorithm [18], consists in modifying the filtering operator of the **P** component such that `filter(P)` computes the arc-consistent closure of P .

Another instructive example is Tabu search [3]. A *Tabu Search* is characterized by the size K of the *Tabu* list that defines the past positions that are declared as *tabu* and to which the algorithm is not allowed to get back. The `forget` and `record` operators correspond to the management of the *Tabu* list. The operator `filter(P)` always returns P (*i.e.* there is no filtering), and the `repair` operators generates a neighbor of the current position in the search space.

Those specializations are given in details in [15]. In that paper, a taxonomy of search algorithms is introduced and several others well known algorithms are presented as specializations of the PLM generic algorithm like:

- systematic algorithms: conflict-based backjumping, `dbt`, `mac-dbt`, etc.
- non-systematic algorithms: `gsat` [19], etc.

Nevertheless, the main interest of the PLM generic algorithm is to be a guide to design new

algorithms. One of the new specialization has been quite successful and is summarized in the following section.

4 Combining CP and LS: the Decision Repair algorithm

The PLM generic algorithm has been also used to design new algorithms: the **decision-repair** family [14]. The idea of **decision-repair** is to combine the propagation-based nature of **mac-dbt** and the true freedom (in the search space exploration) given by a local search algorithm such as **tabu search**.

In terms of the PLM generic algorithm, we have:

- the starting set of decision constraints is empty;
- **filter** uses standard filtering algorithm for reducing the domain of the variables of the problem;
- **record** computes an explanation for the current contradiction and stores it in a *tabu* list² of fixed size K ;
- **forget** erases the oldest stored explanation if the *tabu* list is full;
- **extend** classically adds new decisions (variable assignment, domain splitting, etc.) as long as no solution has been found yet;
- **repair** heuristically selects a decision to undo from the last computed explanations (and whose negation is compatible with the stored explanations).

As several parameters remain unprecised (the way of handling the *tabu* list, the heuristic to be used to select decisions to undo, etc.), **decision-repair** represents a family of algorithms. However, the two main points of that algorithm are: it makes use of a repair algorithm (local search) as a basis, and it works on a partial instantiation in order to be able to use filtering techniques.

The interface between these two worlds, local search and filtering techniques, is built on the good old concept of conflict or explanation. This concept comes from ATMS and is now quite fruitful in constraint programming [11]. Indeed, explanations represent a tractable trace of the solver behavior that can be efficiently used to identify subsets of decisions responsible for dead-ends (providing an efficient **L** component) and to perform dynamic reparations in any constraints system (providing an efficient **M** component), etc. Explanations are now available in some advanced constraint solvers like ILOG Solver [9, 10] and the PALM system [11] (www.e-constraints.net). Moreover, the PALM system provides an implementation of the generic PLM algorithm.

²Notice that this storing structure could be something else than a *tabu* list.

5 Applications

We used the `decision-repair` family algorithm to solve some highly combinatorial scheduling problems.

Classical scheduling shop problems for which a set J of n jobs consisting each of m tasks (operations) must be scheduled on a set M of m machines can be considered as CSP³. One of those problems is called the open-shop problem [4]. For that problem, operations for a given job may be sequenced as wanted but only one at a time. We will consider here the building of non preemptive schedules of minimal makespan⁴. The open-shop scheduling problem is NP-hard as soon as $\min(n, m) \geq 3$. This problem although quite simple to enunciate is really hard to solve optimally: instances of size 6×6 (*i.e.*, 36) variables remain unsolved ! Various heuristics have been proposed⁵: greedy heuristics such as specific list heuristics [6] or local searches such as highly specialized tabu searches [1, 16] or genetic algorithms [17], etc.

We report in table 1 our results on a series of 80 problems (8 series of 10 problems of size 3×3 to 10×10) that have been generated using results presented in [7] for generating really hard open-shop instances (the GP series).

Series	BB-G00	GA-P99	DR	Open instances	DR yield
3×3	10 / 10	10 / 10	10 / 10	0	-
4×4	10 / 10	10 / 10	10 / 10	0	-
5×5	10 / 10	8 / 8	10 / 10	0	-
6×6	9 / 7	2 / 1	10 / 8	3	1 / 1
7×7	3 / 1	6 / 3	10 / 4	9	1 / 3
8×8	2 / 1	2 / 1	10 / 4	9	3 / 7
9×9	1 / 1	0 / 0	10 / 2	9	1 / 9
10×10	0 / 0	5 / 0	5 / 0	10	0 / 5

Table 1: **Results on the GP series.** Results are presented according to the following format: “number of problems solved giving the best results” / “number of optimally solved problems”. **BB-G00** reports the results of an intelligent backtracker described in [5] and stopped after 350 000 backtracks (which represents around 24 hours of cpu time). **GA-P99** reports the results of a genetic algorithm described in [17]. What `decision-repair` gave to the solving of those problem (DR yield) is indicated by “the number of closed instances” / “the number of newly improved instances”

For this set of problems, `decision-repair` (DR in the results) shows all the interest of combining local search and constraint propagation: `decision-repair` closed⁶ 6 of these instances. Furthermore, it provided new best results for 19 other instances; thus it improved known results for 25 instances out of 40 open ones.

Up to size 9×9 , `decision-repair` gives far better results than both the genetic algorithm and branch and bound search (that has been truncated by a time criterion). For 10×10 problems, `decision-repair` is still better than the branch and bound but is matched by the genetic algorithm.

³The variables of the CSP are the starting date of the tasks. Bounds thus represent the least feasible starting time and the least feasible ending time of the associated task.

⁴Ending time of the last task.

⁵We mention here heuristics that are considered to be the best techniques to solve open-shop scheduling problems.

⁶An optimal solution was found and proved – a lower bound is known – for the first time.

6 Conclusion

We introduced in this paper a general algorithm for solving CSP and its application to design a new family of heuristics techniques combining constraint programming and local search: the **decision-repair** family. Our first results were quite surprising because, unlike other specialized algorithms, our implementation remains general and does not need any tuning of complex parameters. **decision-repair**, thanks to its use of explanations, can identify independent sub-problems and stay in a sub-problem until it has been solved. We are currently investigating the impact of the various parameters of the **decision-repair** family and new applications of this new algorithm.

References

- [1] David Alcaide, Joaquín Sicilia, and Daniele Vigo. A tabu search algorithm for the open shop problem. *TOP : Trabajos de Investigación Operativa*, 5(2):283–296, 1997.
- [2] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [3] F. Glover and M. Laguna. *Modern heuristic Techniques for Combinatorial Problems, chapter Tabu Search*, C. Reeves. Blackwell Scientific Publishing, 1993.
- [4] T. Gonzales and S. Sahni. Open-shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4):665–679, 1976.
- [5] Christelle Guéret, Narendra Jussien, and Christian Prins. Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. *European Journal of Operational Research*, 127(2):344–354, 2000.
- [6] Christelle Guéret and Christian Prins. Classical and new heuristics for the open-shop problem. *European Journal of Operations Research*, 107(2):306–314, 1998.
- [7] Christelle Guéret and Christian Prins. A new lower bound for the open-shop problem. *AOR (Annals of Operations Research)*, 92:165–183, 1999.
- [8] William Harvey and Matthew Ginsberg. Limited discrepancy search. In Chris Mellish, editor, *IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence*, Montreal, August 1995.
- [9] ILOG. *Solver 6.0, JSolver 2.1 – Reference Manuals*. 2003.
- [10] Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, Seattle, WA, USA, August 2001.
- [11] Narendra Jussien. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus, 1 December 2001.

Kyoto, Japan, August 25–28, 2003

- [12] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, pages 249–261, Singapore, September 2000. Springer-Verlag.
- [13] Narendra Jussien and Olivier Lhomme. Dynamic domain splitting for numeric CSP. In *European Conference on Artificial Intelligence*, pages 224–228, Brighton, United Kingdom, August 1998.
- [14] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
- [15] Narendra Jussien and Olivier Lhomme. Unifying search algorithms for CSP. Research Report 02-3-INFO, École des Mines de Nantes, Nantes, France, 2002.
- [16] Ching-Fang Liaw. A tabu search algorithm for the open shop scheduling problem. *Computers and Operations Research*, 26, 1998.
- [17] Christian Prins. Competitive genetic algorithms for the open shop scheduling problem. *Mathematical Methods of Operations Research*, 52(3):389–411, 2000.
- [18] Daniel Sabin and Eugene Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP’94: Second International Workshop, Orcas Island, Seattle, USA).
- [19] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *AAAI-92: Proceedings 10th National Conference on AI*, pages 440–446, San Jose, July 1992.