LOCAL SEARCH AS A FIXED POINT OF FUNCTIONS

Eric Monfroy, Frédéric Saubion
University of Nantes, University of Angers, France

Broderick Crawford, Carlos Castro
Pontificia Universidad Católica de Valparaíso, Universidad Técnica Federico Santa María, Valparaíso, Chile

Keywords: Constraint Satisfaction Problems (CSP), Local Search, Constraint Solving.

Abstract: Constraint Satisfaction Problems (CSP) provide a general framework for modeling many practical applications (planning, scheduling, time tabling, ...). CSPs can be solved with complete methods (e.g., constraint propagation), or incomplete methods (e.g., local search). Although there are some frameworks to formalize constraint propagation, there are only few studies of theoretical frameworks for local search. Here we are concerned with the design and use of a generic framework to model local search as the computation of a fixed point of functions.

1 INTRODUCTION

A Constraint Satisfaction Problems (CSP) (Tsang, 1993) is usually defined by a set of variables associated to domains of possible values and by a set of constraints. We only consider here CSP over finite domains. Constraints can be understood as relations over some variables and therefore, solving a CSP consists in finding tuples that belong to each constraint (an assignment of values to the variables that satisfies these constraints).

From a practical point of view, CSP can be solved by using either complete (such as constraint propagation (Apt, 2003)) or incomplete techniques (such as local search (Aarts and Lenstra, 1997) or genetic algorithms (Holland, 1975)). Therefore, constraint solvers mainly rely on the implementation and combination of these techniques. Local search techniques (Aarts and Lenstra, 1997) have been successfully applied to various combinatorial optimization problems.

In the CSP solving context, local search algorithms are used either as the main resolution technique or in cooperation with other resolution processes (e.g., constraint propagation) (Focacci et al., 2002; Jussien and Lhomme, 2002). Unfortunately, the definitions and the behaviors of these algorithms are often strongly related to specific implementations and problems. Our purpose is to use a framework based on functions to provide uniform modeling tools which could help better understanding local search algorithms and designing new ones.

From a more conceptual and theoretical point of view, K.R. Apt has proposed a mathematical framework (Apt, 1997; Apt, 2003) for iteration of a finite set of functions over "abstract" domains with partial ordering: this is well-suited for solving CSPs with constraint propagation.

To obtain a finer definition of local search, in (Monfroy et al., 2008) we proposed a computation structure (the domain of Apt's iterations) which is better suited for local search. We defined the basic functions that can be used iteratively on this structure to create a local search process. We identified the basic processes used for intensification and diversification (move and neighborhood computation) and the process for jumping to other parts of the search space (restart). These three processes are abstracted at the same level by some homogeneous functions called reduction functions. The result of local search is then computed as a fixed point of this set of functions. Possible uses of this framework are illustrated through the description of existing strategies such as descent algorithms (WalkSat) and tabu search (Jaumard et al., 1996).

2 SOLVING CSP WITH LOCAL SEARCH

A CSP is a tuple (X,D,C) where $X = \{x_1,\cdots,x_n\}$ is a set of variables taking their values in their respective domains $D = \{D_1,\cdots,D_n\}$. A constraint $c \in C$ is a relation $c \subseteq D_1 \times \cdots \times D_n$ [1]. In order to simplify notations, D will also denote the Cartesian product of D_i and C the union of its constraints. A tuple $d \in D$ is a solution of a CSP (X,D,C) if and only if $\forall c \in C, d \in c$. In this paper, we always consider D finite.

For the resolution of a CSP (X,D,C), the search space can be often defined as the set of possible tuples of $D = D_1 \times \cdots \times D_n$ and the neighborhood is a mapping $\mathcal{N} : D \to 2^D$.

This neighborhood function defines indeed the possible moves from a sample (i.e., a tuple in this case) to one of its neighbors and therefore fully defines the exploration landscape. The fitness (or evaluation) function *eval* is related to the notion of solution and can be defined as the number of constraints c that are not satisfied by the current sample, i.e., constraints such that $d \notin c$ (d being the currently visited sample, i.e., an element of D in this case). The problem to solve is then a minimization problem. Given a sample $d \in D$, two basic cases can be identified in order to continue the exploration of D: Intensification, choose $d' \in \mathcal{N}(d)$ such that $eval(d') < eval(d)$; Diversification, choose any other neighbor d'. In order to integrate possible restarts (to start new paths) and to generalize this approach we will consider local search as a set of basic local searches.

3 LOCAL SEARCH AS A FIXED POINT OF REDUCTION FUNCTIONS

Local search acts usually on a structure which corresponds to points of the search space. In (Monfroy et al., 2008) we propose a more general and abstract definition based on the notion of sample, already suggested. In our framework, local search is described as a fixed point computation.

[1]For simplicity, we consider that each constraint is over all the variables $x_1, \ldots, x_n$. However, one can consider constraints over some of the x_i. Then, the notion of scheme (Apt, 2003) can be used to denote sequences of variables.

3.1 Chaotic Iterations

K.R. Apt proposed chaotic iterations (Apt, 2003), a general theoretical framework for computing limits of iterations of a finite set of functions over a partially ordered set. In this paper, we do not recall all the theoretical results of K.R. Apt, but we just give the **GI** algorithm for computing fixed point of functions. Consider a finite set F of functions, and d an element of a partially ordered set $\mathcal{D}$.

GI: Generic Iteration Algorithm.

```
$d := \bot$;
$G := F$;
While $G \neq \emptyset$ do
        choose $g \in G$;
        $G := G - \{g\}$;
        $G := G \cup update(G,g,d)$;
        $d := g(d)$;
```

where $\bot$ is the least element of the partial ordering $(\mathcal{D},\sqsubseteq)$, G is the current set of functions still to be applied ($G \subseteq F$), and for all G,g,d the set of functions $update(G,g,d)$ from F is such that:

P1 $\{f \in F - G \mid f(d) = d \wedge f(g(d)) \neq g(d)\} \subseteq update(G,g,d)$.

P2 $g(d) = d$ implies that $update(G,g,d) = \emptyset$.

P3 $g(g(d)) \neq g(d)$ implies that $g \in update(G,g,d)$

Suppose that all functions in F are reduction functions as defined before and that $(\mathcal{D},\sqsubseteq)$ is finite (note that finiteness is important as is has already been mentioned for our structure). Then, every execution of the **GI** algorithm terminates and computes in d the least common fixed point of the functions from F. We now use the **GI** algorithm to compute the fixed point of our functions. The algorithm is thus feed with: a) a set of fair restart functions, fair move and neighborhood functions, that compose the set F; b) initial instantiation of d, and c) the ordering that we use is the ordering $\sqsubseteq$. Unfortunately, the properties P1, P2 and P3 required in (Apt, 2003) for the update functions (to insure the computation of the fixed point) do not fit our extended functions. Indeed, our extended functions are not deterministic and can modify a whole local search even if a previous application did not. This is due to the fact that the select function may choose a configuration which may not be modified by a move (or a neighborhood) function whereas another configuration would be modified by the same move (or neighborhood) function. However, remember the fairness property of the select function: if one configuration can be changed by a move (or neighbor) function, then this configuration will not be neglected forever; and a sample will not be neglected forever by a restart function. Informally, the declarative definition of the update means:

P1 put in the $update(G,g,d)$ all functions not currently in G that will modify $g(d)$. This insures that all effective functions will be re-applied (correctness of the algorithm), whereas ineffective functions will not be added (efficiency reasons).

P2 to ensure termination.

P3 to add g again if it must be re-used.

The update must satisfy the following properties:

P'1 $\{f \in F - G \mid \forall f(d) = d \wedge \exists f(g(d)) \neq g(d)\} \subseteq update(G,g,d)$.

P'2 $\forall g(d) = d$ implies that $update(G,g,d) = \emptyset$

P'3 $\exists g(g(d)) \neq g(d)$ implies that $g \in update(G,g,d)$

Basically, we need to put in the update set of functions, functions that potentially will modify d, the current whole local search. In these conditions, the algorithm terminates and computes the least common fixed point of the functions from F, i.e., the result of the whole local search. Inspired by (Apt, 2003), the proof partially relies on an invariant $\forall f \in F - G, f(d) = d$ of the "while" loop in the algorithm. This invariant is preserved by our characterization of the update function (P'1, P'2 and P'3). Moreover, since we keep a finite partial ordering and a set of monotonic and inflationary functions, the results of K.R. Apt can be extended here.

4 EXPERIMENTATION

The Sudoku problem consists in filling a 9×9 grid so that every row, every column, and every 3×3 box contains the digits from 1 to 9. Although Sudoku, when generalized to n^2 x n^2 grids to be filled in by numbers from 1 to n^2 is NP-complete, the popular 9×9 grid with 3×3 regions is not difficult to solve with a computer program. Therefore, in order to increase the difficulty, we consider 16×16 grids (published under the name "super Sudoku"), 25×25 and 36×36 grids. On this problem, we will show that our generic framework allows us to easily define local search algorithms and to combine and compare them.

4.1 CSP Model

Consider a n^2 x n^2 problem, an instinctive formalization consider a set of n^4 variables whose correspond, to all the cells to fill in. Using this, the set of related constraints is defined by AllDiff global contraints representing: a) all digits appears only once in each row, b) once in each column and c) once in each $n \times n$ square the grid has been subdivided. Concerning LS methods we focus on Tabu search (TS) (Glover and Laguna, 1997). Basically, this algorithm forbids moving to a sample that was visited less than l steps before. To this end, the list of the last l visited samples is memorized. On the other hand, we consider a basic descent technique with random walks (RW) where random moves are performed according to a certain probability p. According to our model, we only have now to design functions of the generic algorithm of Section 3.1 to model strategies. Neighbodhood functions are functions $C \rightarrow C$ such that $(p,V) \mapsto (p, V \cup V')$ with different conditions:

$$\begin{aligned}
FullNeighbor: &\quad V' = \{s \in D | s \notin V\} \\
TabuNeighbor: &\quad V' = \{s \in D | \not\exists k, \\
&\quad n - l \leq k \leq n, s_k = s\} \\
DescentNeighbor: &\quad p = (s_1, \ldots, s_n) \text{ and} \\
&\quad V' = s \subset D \text{ s.t. } \not\exists s' \in V \\
&\quad \text{s.t. } eval(s') < eval(s_n)
\end{aligned}$$

Move functions are functions $C \rightarrow C$ s.t. $(p,V) \mapsto (p', \emptyset)$ with various conditions:

$$\begin{aligned}
BestMove: &\quad p' = p \oplus s' \text{ and} \\
&\quad eval(s') = min_{s'' \in V} eval(s''). \\
ImproveMove: &\quad p = p'' \oplus s_n \text{ and} \\
&\quad p' = p \oplus s \text{ s.t. } eval(s') < eval(s_n). \\
RandomMove: &\quad p' = p \oplus s' \text{ and } s' \in V.
\end{aligned}$$

We can precise here the input set of function F for algorithm GI:

$$\begin{aligned}
Tabusearch: &\quad \{TabuNeighbor \\
&\quad ; BestNeighBor\} \\
Randomwalk: &\quad \{FullNeighbor \\
&\quad ; BestNeighBor; \\
&\quad RandomNeighbor\} \\
TabuSearch + Descent: &\quad \{TabuNeighbor \\
&\quad ; DescentNeighbor \\
&\quad ; ImproveNeighBor \\
&\quad ; BestNeighBor\} \\
Randomwalk + Descent: &\quad \{FullNeighbor \\
&\quad ; BestNeighBor \\
&\quad ; RandomNeighbor \\
&\quad ; DescentNeighbor \\
&\quad ; ImproveNeighBor\}
\end{aligned}$$

The different algorithms correspond to different sets of input functions and different behaviours of the *choose* function in the GI algorithm. *choose* alternatively selects neighborhood and move functions. For the Random Walk algorithm, given a probability parameter p, we have to introduce a quota of p *BestMove* functions and $1 - p$ *RandomMove*

	TabuSearch		
n^2 x n^2	16x16	25x25	36x36
cpu time Avg	3,14	115,08	3289,8
deviations	1,28	52,3	1347,4
mvts Avg	405	3240	22333
	RandomWalk		
n^2 x n^2	16x16	25x25	36x36
cpu time Avg	3,92	105,22	2495
deviations	1,47	49,3	1099
mvts Avg	443	2318	13975
	Descent + TabuSearch		
n^2 x n^2	16x16	25x25	36x36
cpu time Avg	2,34	111,81	
deviations	1,42	55,04	
mvts Avg	534	3666	
	Descent +RandomWalk		
n^2 x n^2	16x16	25x25	36x36
cpu time Avg	2,41	82,94	2455
deviations	1,11	36,99	1092
mvts Avg	544	2581	14908

Figure 1: Results of Sudoku problem by different search approaches.

used in GI. Concerning Tabu Search, we use here a *TabuNeighbor* with $l = 10$ and *BestMove* functions to built our Tabu Search algorithm. At last, we combine a descent strategy by adding *DescentNeighbor* and *ImproveMove* to the previous sets in order to design algorithms in which a Descent is first applied in order to reach more quickly a good configuration.

4.2 Experimentation Results

In Fig. 1 we compare results of the tabu search and random walks associated with descent on different instances of Sudoku problem. We have evaluated the difficulty of the problem with a classic complete method (propagation and split): we obtained a more than one day cpu time cost for a 36×36 grid. At the opposite, by a simple formalization of the problem and thanks to a function application model, we are able to reach a solution with classical local search algorithms starting from an empty grid. For each method and for each instance, 2000 runs were performed (except for 36×36 problem, 500 runs). Adding descent in a Tabu Search or in a Random Walk method, allows us to reduce the computation time to reach a solution. The hybrid strategies combining several move and neighborhood functions provide better results.

5 CONCLUSIONS

In this paper, we have used a framework to model local search as a fixed point of functions for solving Sudoku. This framework provides a computational model inspired in the initial works of K.R. Apt (Apt, 2003). It helps us to finer define the basic processes of local search at a uniform description level and to describe specific search strategies. This mathematical framework could be helpful for the design of new local search algorithms, the improvement of existing ones and their combinations. Our framework could also be used for experimental studies as it provides a uniform description framework for various methods in an hybridization context.

REFERENCES

Aarts, E. and Lenstra, J. K., editors (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA.

Apt, K. R. (1997). From chaotic iteration to constraint propagation. In Degano, P., Gorrieri, R., and Marchetti-Spaccamela, A., editors, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 36–55. Springer.

Apt, K. R. (2003). *Principles of Constraint Programming*. Cambridge Univ. Press.

Focacci, F., Laburthe, F., and Lodi, A., editors (2002). *Local Search and Constraint Programming. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Norwell, MA.

Glover, F. and Laguna, F. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Jaumard, B., Stan, M., and Desrosiers, J. (1996). Tabu search and a quadratic relaxation for the satisfiability problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:457–478.

Jussien, N. and Lhomme, O. (2002). Local search with constraint propagation and conflict-based heuristics. *Artif. Intell.*, 139(1):21–45.

Monfroy, E., Saubion, F., Crawford, B., and Castro, C. (2008). Towards a formalization of combinatorial local search. In *Proceedings of the International MultiConference of Engineers and Computer Scientists, IMECS, March 19-21, 2008, Hong Kong, China*, Lecture Notes in Engineering and Computer Science. Newswood Limited.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London.