Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 637809, 10 pages
http://dx.doi.org/10.1155/2015/637809

Research Article

A Variable Depth Search Algorithm for Binary Constraint Satisfaction Problems

N. Bouhmala

Department of Technology and Maritime Innovation, Buskerud and Vestfold University College, P.O. Box 4, 3199 Borre, Norway

Correspondence should be addressed to N. Bouhmala; noureddine.bouhmala@hbv.no

Received 7 October 2014; Revised 5 March 2015; Accepted 1 April 2015

Academic Editor: Jianming Shi

The constraint satisfaction problem (CSP) is a popular used paradigm to model a wide spectrum of optimization problems in artificial intelligence. This paper presents a fast metaheuristic for solving binary constraint satisfaction problems. The method can be classified as a variable depth search metaheuristic combining a greedy local search using a self-adaptive weighting strategy on the constraint weights. Several metaheuristics have been developed in the past using various penalty weight mechanisms on the constraints. What distinguishes the proposed metaheuristic from those developed in the past is the update of k variables during each iteration when moving from one assignment of values to another. The benchmark is based on hard random constraint satisfaction problems enjoying several features that make them of a great theoretical and practical interest. The results show that the proposed metaheuristic is capable of solving hard unsolved problems that still remain a challenge for both complete and incomplete methods. In addition, the proposed metaheuristic is remarkably faster than all existing solvers when tested on previously solved instances. Finally, its distinctive feature contrary to other metaheuristics is the absence of parameter tuning making it highly suitable in practical scenarios.

1. Introduction

Organizations like companies or public institutions are confronted in their daily life with a large number of combinatorial optimization problems which occur in many different application domains such as Operations Research (e.g., scheduling and assignment), hardware design (verification and testing, placement and layout), financial decision making (option trading or portfolio management), or even biology (DNA sequencing). The domain of combinatorial optimization refers to optimization problems where the search space (i.e., the set of all feasible solutions) is discrete. The constraint satisfaction problem (CSP) which can model a wide spectrum of combinatorial optimization problems rising in the field of artificial intelligence has become an important field of study in both theoretical and applied computer science. Constraint technology is making a considerable commercial impact worldwide due to its ability to solve highly complex applications operating in the most difficult environment counting on first-class technology to perform the job. ILOG and Cosytec are two of the leading companies producing software based on this technology. A large number of systems based on the constraints technology have been developed. Examples include the APACHE system [1] used at Roissy Airport in Paris, PLAN system [2] which is a medium-long term scheduling system for aircraft assembly line scheduling, the COBRA system [3] that generates work plans for train drivers and conductors of North Western Trains in the UK, and TAP-AI which is a planning system for crew assignment in the airline SAS [4]. Disasters which have long impacted world nations, resulting in mass casualties and huge financial tolls where technology and humans have to work together hand-in-hand without fault, with every single step of a mission meticulously planned out, are another research area where solutions based on constraint technology have received a great attention lately [5, 6]. The handbook of Constraint Programming [7] lists example applications of several areas modeled as CSPs. The paper is organized as follows. Section 2 explains the constraint satisfaction problem. Section 3 provides a survey of methods used to solve the constraint satisfaction problem. Section 4 introduces the metaheuristic in detail. Section 5 presents the results while Section 6 concludes the paper.

2. CSP

The CSP consists of assigning values to variables while satisfying certain constraints. Constraints can be given explicitly, by listing all possible tuples or implicitly, by describing a relation in some mathematical form. As a domain example, consider problems that occur in production scheduling. Scheduling is concerned with the allocation of resources to activities with the goal of optimizing some performance objectives while satisfying certain restrictions or constraints. Depending on the problem posed, resources may refer to machines, humans, and so forth, activities could be manufacturing operations, objectives could be the minimization of the schedule length, and finally constraints may state the precedence relationship among activities as they govern the schedule solution.

A CSP is a tuple $\langle X, D, C\rangle$, where

(i) X is a finite set of variables: $X = \{X_1, X_2, \ldots, X_n\}$,

(ii) D is a finite set of domains: $D = \{D_{X_1}, D_{X_2}, \ldots, D_{X_n}\}$. Thus each variable $X_i \in X$ has a corresponding discrete domain D_{X_i} from which it can be instantiated,

(iii) $C = \{C_1, C_2, \ldots, C_k\}$ is a finite set of constraints. Each k-ary constraint restricts a k-tuple of variables $(X_1, X_2, \ldots, X_k)$ and specifies a subset of $D_1 \times \cdots \times D_k$, each element of which is values that the variables can not take simultaneously. This set is referred to as the no-good set (i.e., an assignment set that is not contained in any solution.)

A solution to a CSP requires the assignment of values to each of the variables from their domains such that all the constraints on the variables are satisfied. In this paper, attention is focused on binary CSPs, where all constraints are binary; that is, they are based on the Cartesian product of the domains of two variables. However, any nonbinary CSP can theoretically be converted to a binary CSP [8, 9]. The structure of a binary CSP can be better visualized by a graph $G(V, E)$ where the set of vertices V corresponds to the variables and each edge $(X_i, X_j) \in E$ represents a constraint connecting the pair of variables involved in this constraint. The CSP in its general form is NP-complete [10] and has been extensively studied due to its simplicity and applicability [7]. The simplicity of the problem coupled with its intractability makes it an ideal platform for exploring new algorithmic techniques. This has led to the development of several algorithms for solving CSPs which usually fall into two main categories: systematic algorithms and local search algorithms.

3. A Brief Survey of Methods

Systematic search algorithms rely on a systematic way in their exploration of the search space. These methods [11–16] aim at exploring the entire solution space using tree search algorithms. The two main components of a tree search are the way to go forward, that is, which decision is taken at which point of the search and the way to go backwards, that is, the backtracking strategy that defines how the algorithm will behave when an inconsistency is detected. In practice, methods based on systematic tree search may fail to solve large and complex CSPs instances because the computing time required may become prohibitive. For instance, a CSP with n variables, each with a domain of size m, makes the search space which is to be explored proportional to $O(m^n)$, that is, exponential in the number of variables. Most searches that come up in CSPs occur over spaces that are far too large to be searched exhaustively. One way to overcome the combinatorial explosion is to give up completeness. Stochastic local search (SLS) algorithms are techniques which use this strategy and gained popularity due to their conceptual simplicity and good performance. These methods start with an initial assignment of values to variables randomly or heuristically generated. During each iteration, a new solution is selected from the neighborhood of the current one by performing a move. A move might consist in changing the value of one randomly selected variable. Choosing a good neighborhood and a method for searching it is usually guided by intuition, because very little theory is available as a guide. If the new solution provides a better value in light of the objective function, the new solution becomes the current one. In order to avoid premature convergence, SLS methods resort to some sort of randomization (noise probability) to avoid local minima and to better explore the search space. The search is iterated until a termination criterion is reached. Most algorithms applied to CSPs use the so-called 1-exchange neighborhood under which two solutions are direct neighbors if, and only if, they differ at most in the value assigned to one variable. A basis for many SLS algorithms is the minimum conflict heuristic MCH [17]. MCH iteratively modifies the assignment of a single variable in order to minimize the number of violated constraints. Since the introduction of MCH there have been a large number of local search heuristics proposed to tackle CSPs. Several representative state-of-the-art SLS in the literature include the break method for escaping from local minima [18], various enhanced MCH (e.g., randomized iterative improvement of MCH called WMCH [19], MCH with tabu search [20, 21]), and a large body of work on evolutionary algorithms for CSPs [22–26] for interested readers. Weights-based algorithms have been advocated by the intuition that, by introducing weights on variables or constraints, local minima can be avoided and the search process can learn to distinguish between critical and less critical constraints. Methods belonging to this category include genet [27], guided local search [28], discrete Lagrangian search [29], the exponentiated subgradient [30], the scaling and probabilistic smoothing [31], evolutionary algorithms combined with stepwise adaptation of weights [32–34], methods based on dynamically adapting weights on variables [35], or both (i.e., variables and constraints) [36]. Weighting schemes have been also combined with systematic methods to reduce the size of tree search methods and consequently speeding up the solving time [37–39]. Recently, an improved version of the Squeaky Wheel Optimization (SWO) [40] originated in [41] has been proposed for the scheduling problem. In SWO, a greedy algorithm is used to construct an initial solution which is then analyzed in order to identify those tasks that if improved are likely to improve the objective function score. The improved version provides additional postprocessing transformations to explore the neighborhood enhanced with a stochastic local search algorithm. Methods based on large neighborhood search have recently attracted several researchers for solving the CSP [42]. The central idea is to reduce the size of local search space relying on a continual relaxation (removing elements from the solution) and reoptimization (reinserting the removed elements). Systematic methods exhibit poor performance on large problems because bad decisions made early in the search persist for exponentially long times. In contrast, stochastic local search methods replace systematicity with stochastic techniques for diversifying the search. However, the lack of systematicity makes remembering the history of past states problematic. To this end, hybrid search methods offering desirable aspects of both systematic methods and local search methods are becoming more and more popular and interested readers may refer to [43–45] to get a deeper understanding on these mixed methods.

4. Variable Depth Search Algorithm

Traditional local search algorithms for solving CSP problems start from an initial solution s and repeat replacing s with a better solution in its neighborhood $N(s)$ until no better solution is found in $N(s)$, where $N(s)$ is a set of solutions obtained from s by updating the value of one selected variable. A solution $s*$ is called locally optimal if no better solution exists in $N(s*)$. The algorithm proposed in this paper belongs to the class of variable depth search algorithms where an existing solution is not modified just by making a change to a single variable; instead, the changes affect as many variables as possible when moving from one solution to another. The algorithm is inspired from the famous Kerninghan-Lin algorithm used for solving the graph partitioning problem [46] and the traveling salesman problem [47]. The idea is to replace the search for one favorable move (i.e., the update of one variable) by a search for a favorable sequence of moves (i.e., the update of a series of variables) using the criterion of score to guide the search. The different steps of the algorithm are described in Algorithm 1.

(i) Random-initial-solution (): the algorithm starts building an initial solution. The initial solution is simply constructed by assigning to each variable X_i a random value v_i from D_{X_i} (Line 5 of Algorithm 1). Based on these values, the status of each constraint is set to either violated or nonviolated.

(ii) Assign-Initial-Weights (): during this step the algorithm assigns a fixed amount of weight equal to 1 across all the constraints (Line 6 of Algorithm 1). The distribution of weights to constraints is a key factor to the success of the algorithm. During the course of the search, the algorithm forces hard constraints (i.e., those with large weights) to be satisfied thereby preventing the algorithm at a later stage from getting stuck at a local optimum.

(iii) Stopping criterion: the outer loop (Line 7 of Algorithm 1) determines the stopping criterion met by the algorithm. The algorithm stops if a solution has been found (i.e., all the constraints are satisfied) or if a time limit has been reached.

(iv) Random-selected-variable (): a starting random variable from which the searching process begins is selected and added to the set T (Lines: 9, 10, and 11 of Algorithm 1).

(v) Inner loop: the inner loop (Lines: 12, 13, 14, 15, 16, 17, and 18 of Algorithm 1) proceeds by repeatedly selecting for each variable X_i removed from the set T, the value v_{best} from its domain D_{X_i} producing the highest score. Given the choice between several equally high scores, the algorithm picks one value at random. The score of a variable $X_i^{v_j}$ is defined as the increase (or decrease in the number of violated constraints) in the number of nonviolated constraints if X_i is assigned the value v_j. The score is given by

$$\text{Score}\left(X_i^{v_{\text{best}}}\right) = \text{New}\left(X_i^{v_{\text{best}}}\right) - \text{Current}\left(X_i^{v_{\text{current}}}\right), \quad (1)$$

$$\begin{aligned} &\text{New}\left(X_i^{v_{\text{best}}}\right) \\ &\quad = \sum_{X_j \in \text{Neigh}(X_i)}^{|\text{Neigh}(X_i)|} \Omega\left(X_i, X_j\right) * \Phi\left(X_i^{v_{\text{best}}}, X_j^{v_{\text{current}}}\right), \\ &\text{Current}\left(X_i^{v_{\text{current}}}\right) \\ &\quad = \sum_{X_j \in \text{Neigh}(X_i)}^{|\text{Neigh}(X_i)|} \Omega\left(X_i, X_j\right) * \Phi\left(X_i^{v_{\text{current}}}, X_j^{v_{\text{current}}}\right). \end{aligned} \quad (2)$$

Equations (2) calculates the sum of the weights of the satisfied constraints the variable X_i is involved with. $\Omega(X_i, X_j)$ denotes the weight of the constraint connecting X_i and X_j while the second term returns the value of 1 if the constraint is satisfied and 0 otherwise. Thus, after the selection of v_{best} and inserting $X_i^{v_{\text{best}}}$ into the set M_{Best}, the status (i.e., violated or nonviolated) of the constraints for the neighboring variables of X_i is updated. Consider the following:

(i) Highest cumulative score: an iteration of the algorithm terminates when the set T becomes empty. In this way, a sequence of scores with corresponding variables and their selected values is formed. Thereafter, the algorithm identifies the subset of variables having the highest cumulative score (HCS) (Line 19 of Algorithm 1). The identification of this subset is equivalent to choosing k so that HCS(k) in (3) is maximum, where $S_{X_i}^{v_{\text{best}}}$ represents the score of the variable X_i corresponding to the value v_{best}. Finding k is the same as solving the maximum subarray problem introduced for the first time in [48]. The problem is usually solved using Kadane's algorithm [49] which simply accumulates a partial sum and

```
input: Problem Instance
output: Number of satisfied constraints
(1) begin
(2)     Let $Neigh(X_i) = \{X_j \mid (X_i, X_j) \in E, i = 1, \ldots, n, j = 1, \ldots, n\}$;
(3)     Let $X_i^{v_j}$ denotes the assignment of the value $v_i$ from $D_{X_i}$ to $X_i$;
(4)     Let $M_{\text{Best}} = \{X_i^{v_{\text{best}}} \mid Score(X_i^{v_{\text{best}}}) >= Score(X_i^{v_j}), i = 1, \ldots, n, j = 1, \ldots, |D_{X_i}|\}$
(5)     Random-Initial-Solution ();
(6)     Assign-Initial-Weights ();
(7)     while (!stop) do
(8)         $M_{\text{Best}} = \emptyset$;
(9)         $T = \emptyset$;
(10)        $X_i \leftarrow$ Random-Selected-Variable ();
(11)        $T \leftarrow T \cup \{X_i\}$;
(12)        while ($T \neq \emptyset$) do
(13)            $T \setminus \{X_k\} \leftarrow$ Remove a random variable $X_k$ from $T$;
(14)            $X_k^{v_{\text{best}}} \leftarrow$ Assign the value $v_{\text{best}}$ to $X_k$ producing the highest score;
(15)            $T \leftarrow T \cup \{X_j | X_j \in Neigh(X_i) \wedge X_j \notin T\}$;
(16)            $M_{\text{Best}} \leftarrow M_{\text{Best}} \cup \{X_k^{v_{\text{best}}}\}$;
(17)            Update-Score of Neigh ($X_k$);
(18)        end
(19)        Identify the set of variables with the highest cumulative score (HCS):
(20)        $\text{HCS}(k) = \sum_{i=1, X_i^{v_{\text{best}}} \in M_{\text{best}}}^{k} (S_{X_i^{v_{\text{best}}}})$;
(21)        if ($\text{HCS}(k) \geq 0$) then
(22)            Assign all the variables up to the index $k$ with their new best values;
(23)        else
(24)            Assign the variable at the index 1 with its new best value;
(25)        end
(26)        Adjust-Weights ();
(27)    end
(28) end
```

Algorithm 1: VNS-CSP.

updates the optimal range when this partial sum becomes larger than the global sum. If HCS ≥ 0, the solution is updated by substituting all the variables up to the index k with their new values; otherwise the update is restricted to just the first variable (index 1) (Lines: 20, 21, 22, 23, and 24 of Algorithm 1):

$$\text{HCS}(k) = \sum_{i=1, X_i^{v_{\text{best}}} \in M_{\text{best}}}^{k} S_{X_i^{v_{\text{best}}}}. \quad (3)$$

(ii) Adjust-Weights: finally, the algorithm proceeds with the weighting process divided into two distinct steps (Line 25 of Algorithm 1). The weights of each newly violated constraint are then increased by one, whereas the newly satisfied constraints will have their weights decreased by one before another round of the algorithm is repeated or the stopping criterion is reached. This weighting procedure is the same as the one adopted in [18].

5. Experimental Results

5.1. Test Instances. The performance of the metaheuristic (VNS-CSP) has been tested on hard random CSP problems taken from Lecoutres benchmark [50] under the name RB-Model. This model enjoys several features that makes it of a great theoretical and practical interest [51]. Tables 1 and 2 show the list of problem instances used in the experiments. The list contains 8 classes of problems each of which is composed of 5 instances, giving a total of 40 instances. Table 1 shows the list of solved hard problems, while Table 2 refers to those problems that remain challenging for most solvers. They are all located in the exact phase transition point [52] and the hardness of solving these instances grows exponentially with the number of variables. The first column denotes the number of variables, the second column the domain size of the each variable, and the third column the number of constraints; the fourth column specifies the combination of values not allowed (no-good) and the last column shows whether the instance has already been solved by an existing solver. All the benchmark instances used in this experiment are satisfiable instances. Each problem instance was run 100 times (i.e., each run is performed with a different seed) with a cut-off parameter (max-time) set to 15 minutes. The tests were carried out on a DELL machine with 800 MHz CPU and 2 GB of memory. The code was written in C and compiled with the GNU C compiler version 4.6.

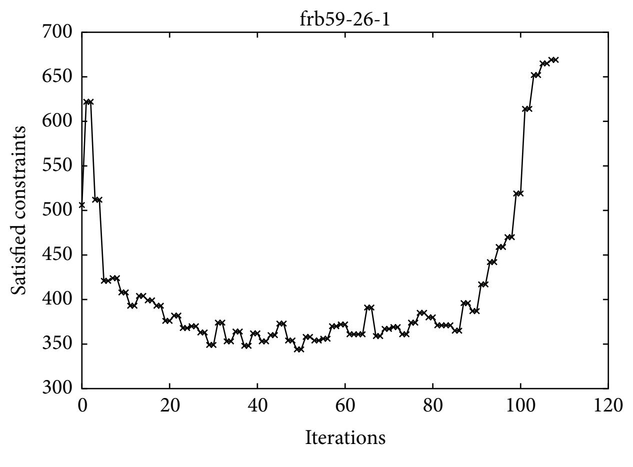
Table 1: Solvable instances.

Instance	Variables	Values	Constraints	No-good	Solved
frb30-15-1.csp	30	15	284	56	Yes
frb30-15-2.csp	30	15	284	56	Yes
frb30-15-3.csp	30	15	284	56	Yes
frb30-15-4.csp	30	15	284	56	Yes
frb30-15-5.csp	30	15	284	56	Yes
frb35-15-1.csp	35	17	346	72	Yes
frb35-15-2.csp	35	17	346	72	Yes
frb35-15-3.csp	35	17	346	72	Yes
frb35-15-4.csp	35	15	346	72	Yes
frb35-15-5.csp	35	15	346	72	Yes
frb40-19-1.csp	40	19	410	90	Yes
frb40-19-2.csp	40	19	410	90	Yes
frb40-19-3.csp	40	19	410	90	Yes
frb40-19-4.csp	40	19	410	90	Yes
frb40-19-5.csp	40	19	410	90	Yes
frb45-21-1.csp	45	21	476	110	Yes
frb45-21-2.csp	45	21	476	110	Yes
frb45-21-3.csp	45	21	476	110	Yes
frb45-21-4.csp	45	21	476	110	Yes
frb45-21-5.csp	45	21	476	110	Yes
frb53-24-3.csp	53	24	585	144	Yes

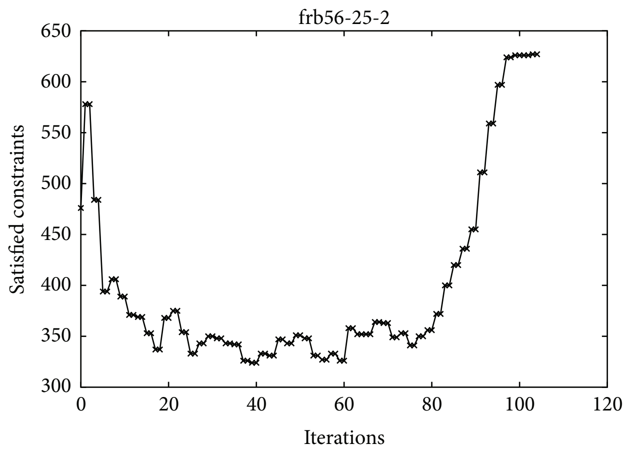
Table 2: Benchmark instances: unsolvable instances.

Instance	Variables	Values	Constraints	No-good	Solved
frb50-23-1.csp	50	23	544	132	**No**
frb50-23-2.csp	50	23	544	132	**No**
frb50-23-3.csp	50	23	544	132	**No**
frb50-23-4.csp	50	23	544	132	**No**
frb50-23-5.csp	50	23	544	132	**No**
frb53-24-1.csp	53	24	585	144	**No**
frb53-24-2.csp	53	24	585	144	**No**
frb53-24-4.csp	53	24	585	144	**No**
frb53-24-5.csp	53	24	585	144	**No**
frb56-25-1.csp	56	25	627	156	**No**
frb56-25-2.csp	56	25	627	156	**No**
frb56-25-3.csp	56	25	627	156	**No**
frb56-25-4.csp	56	25	627	156	**No**
frb56-25-5.csp	56	25	627	156	**No**
frb59-26-1.csp	59	26	669	169	**No**
frb59-26-2.csp	59	26	669	169	**No**
frb59-26-3.csp	59	26	669	169	**No**
frb59-26-4.csp	59	26	669	169	**No**
frb59-26-5.csp	59	26	669	169	**No**

5.2. Algorithm's Behavior. The plots depicted in Figures 1 and 2 show the evolution of the mean satisfied number of constraints as a function of the number of iterations for 4 hard problems that remain difficult for most solvers. These plots have been selected as they represent the general trend observed on all the problem instances. Investigating the trends of the algorithm from the plots suggests the presence of three different distinct phases. The first phase corresponds to the first iteration of the algorithm where all the constraints are assigned a weight equal to 1. This similar weight provides all the constraints with equal chances for being satisfied. In all the studied cases, the curves have a tendency to go uphill showing an improvement in the number of satisfied constraints. The second phase which takes most of the time corresponds to a diversification stage. During this second phase, the weights assigned to various constraints alter after each iteration depending on the status of the constraints (i.e., satisfied or unsatisfied) forcing the algorithm to favor the satisfaction of hard constraints (i.e, constraints with higher weights). This weighting of constraints results in worsening the quality of the solution by falling drastically during early stages of this phase (on average between 33% and 53%) and continues to exhibit a varying increasing decline rate over time before the curves start moving uphill marking the start of the intensification phase. This phase which tends to be of short duration compared to the diversification phase is characterized by the absence of downhill moves. A downhill move occurs when the set of changes determined by the algorithm reduces the number of satisfied constraints. During the intensification phase, the algorithm intensifies the search around promising areas of the search space making the number of satisfied constraints to climb sharply until all the constraints of the problem are satisfied. The termination of the diversification phase ensures that each constraint relating at most two variables is assigned an ideal weight expressing its relative hardness taking into account the values assigned to its relating variables and the values of the variables defining the neighboring constraints. This ideal weight leads the system to enter a state of balance that is required for the intensification phase to be triggered leading the algorithm to easily reach the solution of the problem. Figures 3 and 4 show the evolution of the number of satisfied constraints and the sum of weights of satisfied constraints through the diversification and intensification phases, respectively. Figure 3 reveals that improving the sum of weights of satisfied constraints does not necessarily imply an increase in the number of satisfied constraints. Satisfying constraints with large weights may introduce a new set of unsatisfied constraints leading to a further decrease in the number of satisfied constraints. Another interesting remark to be drawn from this plot is the ability of the algorithm to escape from the so-called plateau regions or local optima. Plateaus represent regions of the search space containing states with only equal or disimproving costs leaving the best solution unchanged. Figure 4 shows a continuous improvement of the two curves during the intensification phase until the solution of the problem is reached. Figure 5 shows the impact of weighting and nonweighting strategies on the algorithm's convergence. The plot illustrates the easiness encountered by the algorithm without the weighting mechanism in improving the number of satisfied constraints during the first iterations of the algorithm (up to 96% of the constraints are satisfied) before getting permanently stuck in long plateau regions or a local maximum leading to a premature convergence due to its greedy bias. The superior performance of the algorithm is

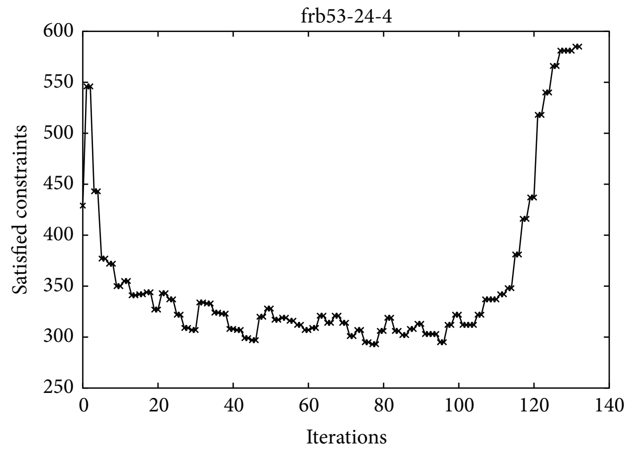


(a)

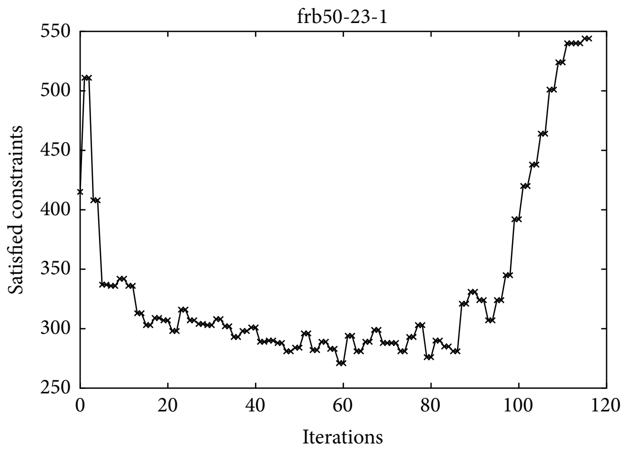


(b)

Figure 1: Evolution of the number of satisfied constraints: (a) frb59-26-1 and (b) frb56-25-2.



(a)



(b)

Figure 2: Evolution of the number of satisfied constraints: (a) frb53-24-4 and (b) frb50-23-1.

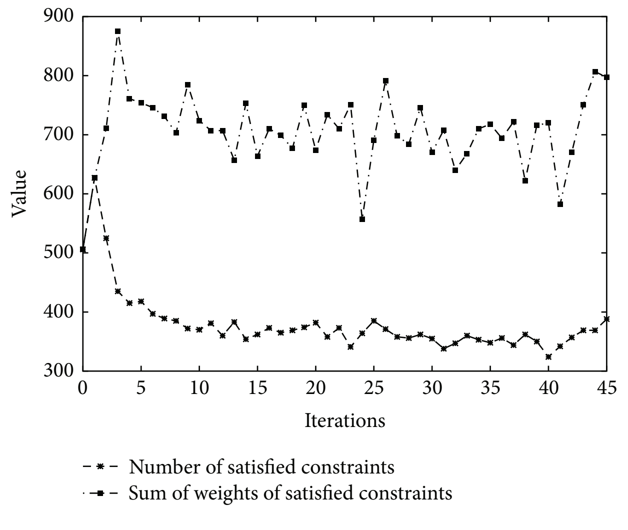


Figure 3: Evolution of the number of satisfied constraints and the sum of weights of satisfied constraints for frb59-26-3 during the diversification phase.

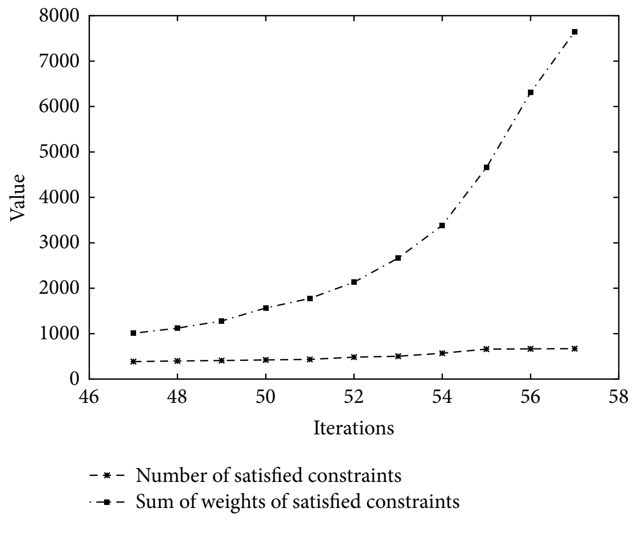


Figure 4: Evolution of the number of satisfied constraints and the sum of weights of satisfied constraints for frb59-26-3 during the intensification phase.

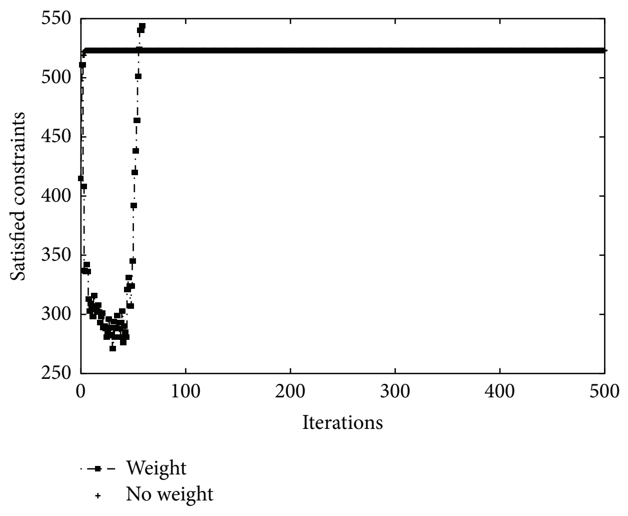


Figure 5: Impact of the nonweight mechanism on the algorithm's convergence for frb50-23-3.

further made evident by looking at Table 3 which presents the results for already solved problem instances and unsolvable problem instances (instances in bold) that still present a challenge for all existing solvers. The results illustrate the performance of the algorithm reflecting its success ratio (i.e., defined as the ratio of successful runs with respect to the total number of runs) and the amount of time taken to reach the solution. From these results, the algorithm has a very good reliability (the success ratio is 100%). In terms of speed, VNS-CSP reaches the solutions in short computational times. Hence much of the difference in the run time (max-min) is due to the different random initial solutions and to the first chosen random variable that initiates the searching process.

5.3. Comparison with State-of-the-Art Solvers. Tables 4–7 compare the time (i.e., the average time over 100 runs) required for different state-of-the-art solvers relative to that required by VNS-CSP. All these solvers are complete (i.e., systematic) solvers. The dash symbol means that the solver could not find the solution after 30 minutes. The first row on each table refers to the metaheuristic proposed in this work VNS-CSP. In all cases, the proposed metaheuristic remains the fastest of them all. The time of the proposed metaheuristic ranges from 10% to 90% of the time of the best solver and from 5 to several hundred times faster than the slowest solver. Table 8 compares VNS-CSP against two variants of Ant Colony Optimization (ACO) algorithms and tabu search. This table is extracted from [53]. The first column shows the 8-class problems each of which is composed of 5 different instances. The second and third columns show the results of the two variants of ACO. The first number represents the number of solved instances, while the number in bracket gives the average CPU time on 3 GHz Intel Xeon. The last column shows the result of VNS-CSP. The time in bracket is the average time taken on DELL machine with 800 MHz CPU. This table is only meant as a rough guide since VNS-CSP and the other algorithms are run on different machines. The table shows that the two variants of ACO and tabu are outperformed by VNS-CSP. VNS-CSP solved all the instances, while ACO-vertex has been able to solve 29 out of 40, ACO-clique 28 out of 40, and tabu 36 out of 40. Comparing the time of the different algorithms, VNS-CSP is the one requiring the least amount of time.

Table 3: Benchmark instances: all unsolved instances solved.

	Execution time and success ratio			
Instances	Min (sec)	Max (sec)	Mean (sec)	Success ratio
frb30-15-1.csp	0.18	0.20	0.19	100%
frb30-15-2.csp	0.15	0.17	0.16	100%
frb30-15-3.csp	0.16	0.17	0.17	100%
frb30-15-4.csp	0.18	0.19	0.19	100%
frb30-15-5.csp	0.14	0.23	0.20	100%
frb35-17-1.csp	0.22	0.27	0.25	100%
frb35-17-2.csp	0.29	0.42	0.31	100%
frb35-17-3.csp	0.21	0.25	0.24	100%
frb35-17-4.csp	0.20	0.23	0.22	100%
frb35-17-5.csp	0.27	0.30	0.28	100%
frb40-19-1.csp	0.37	0.42	0.41	100%
frb40-19-2.csp	0.34	0.38	0.36	100%
frb40-19-3.csp	0.41	0.43	0.42	100%
frb40-19-4.csp	0.36	0.42	0.38	100%
frb40-19-5.csp	0.39	0.41	0.40	100%
frb45-21-1.csp	0.50	0.54	0.52	100%
frb45-21-2.csp	0.52	0.55	0.54	100%
frb45-21-3.csp	0.60	0.66	0.62	100%
frb45-21-4.csp	0.54	0.56	0.54	100%
frb45-21-5.csp	0.53	0.58	0.56	100%
frb50-23-4.csp	0.78	0.86	0.81	100%
frb53-24-2.csp	0.87	0.92	0.89	100%
frb50-23-1.csp	0.72	0.84	0.79	100%
frb50-23-2.csp	0.73	0.76	0.75	100%
frb50-23-3.csp	0.86	0.93	0.88	100%
frb50-23-5.csp	0.77	0.81	0.80	100%
frb53-24-1.csp	0.95	1.01	0.98	100%
frb53-24-3.csp	0.88	0.95	0.87	100%
frb53-24-4.csp	1.05	1.09	1.07	100%
frb53-24-5.csp	1.15	1.18	1.17	100%
frb56-25-1.csp	1.20	1.22	1.21	100%
frb56-25-2.csp	0.99	1.03	1.01	100%
frb56-25-3.csp	1.01	1.05	1.02	100%
frb56-25-4.csp	0.98	0.99	0.99	100%
frb56-25-5.csp	1.09	1.14	1.11	100%
frb59-26-1.csp	1.24	1.33	1.30	100%
frb59-26-2.csp	1.22	1.26	1.24	100%
frb59-26-3.csp	1.31	1.34	1.32	100%
frb59-26-4.csp	1.30	1.32	1.31	100%
frb59-26-5.csp	1.23	1.27	1.24	100%

Table 4: Comparing various solvers: frb30 instances.

Solver	frb30-15-1	frb30-15-3	frb30-15-4	frb30-15-5
VNS-CSP	**0.19**	**0.17**	**0.19**	**0.20**
Abscon112v4	0.91	1.41	1.33	1.65
Abscon 112v4ESAC	0.88	1.36	1.35	1.67
Bpsolver09	0.67	2.49	2.59	1.3
Choco2.1.1	1.48	2.21	2.55	1.58
Choco2.1.1b	1.82	1.14	2.33	0.90
Concrete	1.89	1.25	2.20	1.75
Concrete DC	2.59	2.94	3.30	2.41
Conquer	2.45	2.77	1.00	1.61
Mistral	0.21	0.22	0.26	0.08
pcs	3.05	2.51	1.93	1.07
pcs-restart	4.92	0.55	2.55	0.70
SAT4JCSP	3.74	4.81	5.50	3.95
Sugarv1.14.6 + minisat	3.3	1.17	1.97	1.15
Sugarv1.14.6 + picosat	2.11	1.47	1.81	1.57

Table 5: Comparing various solvers: frb35 instances.

Solver	frb35-17-1	frb35-17-2	frb35-17-4
VNS-CSP	**0.25**	**0.31**	**0.22**
Abscon112v4	4.37	6.71	3.54
Abscon 112v4ESAC	3.92	6.38	3.87
Bpsolver09	16.81	67.03	25.30
Choco2.1.1	4.02	35.90	8.47
Choco2.1.1b	5.40	16.27	1.70
Concrete	5.20	10.86	4.49
Concrete DC	6.68	7.05	5.83
Conquer	2.74	14.73	5.57
Mistral	0.64	3.08	0.25
pcs	42.05	27.39	8.15
pcs-restart	47.99	27.07	1.94
SAT4JCSP	51.20	212.57	43.08
Sugarv1.14.6 + minisat	13.69	25.35	4.17
Sugarv1.14.6 + picosat	14.89	4.72	14.37

6. Conclusions

This paper proposes a variable depth search algorithm for the CSP problem. The heart of the metaheuristic relies on a combination between an adaptive weighting strategy on the constraint weights and a greedy search. This combination proved to be an excellent mechanism to guide the search in order to achieve a suitable trade-off between intensification and diversification. The proposed metaheuristic has been experimentally evaluated on hard random CSP problems belonging to RB-Model. The difficulty of solving some of these problems by state-of-the-art solvers highlights the capabilities of the proposed metaheuristic. Indeed, the experimental results have been very positive, solving all unsolvable instances in very short computational times. Most

Table 6: Comparing various solvers: frb40 instances.

Solver	frb40-19-1	frb40-19-4	frb40-19-5
VNS-CSP	**0.41**	**0.38**	**0.40**
Abscon112v4	1.30	82.98	37.29
Abscon112v4ESAC	1.32	79.60	36.6
Bpsolver09	361.45	137.45	199.04
Choco2.1.1	25.43	24.33	161.97
Choco2.1.1b	25.43	24.33	161.97
Concrete	15.36	87.77	177.15
Concrete DC	21.27	114.80	89.01
Conquer	21.66	9.49	94.78
Mistral	2.15	7.73	63.46
pcs	1758.00	—	—
pcs-restart	—	—	1107.76
SAT4JCSP	272.64	500.71	10.49
Sugarv1.14.6 + minisat	48.30	186.73	—
Sugarv1.14.6 + picosat	41.52	364.37	297.49

Table 7: Comparing various solvers: frb45 and frb53 instances.

Solver	frb45-21-2	frb45-21-4	frb45-21-5	frb53-24-3
VNS-CSP	**0.54**	**0.54**	**0.56**	**0.87**
Abscon112v4	275.31	478.01	1228.41	1342
Abscon112v4ESAC	284.00	441.30	1194.47	245.31
Bpsolver09	—	—	—	—
Choco2.1.1	1305.51	65.05	1635.94	—
Choco2.1.1b	423.62	217.20	89.18	—
Concrete	394.55	330.23	672.39	—
Concrete DC	498.34	359.80	1023.49	—
Conquer	800.71	716.26	878.97	—
Mistral	224.39	66.05	121.84	—
pcs	—	—	—	—
pcs-restart	—	—	—	—
SAT4JCSP	—	—	—	—
Sugarv1.14.6 + minisat	56.95	—	1114.95	—
Sugarv1.14.6 + picosat	795.52	—	—	—

Table 8: Comparing VNS-CSP with tabu and ACO metaheuristics.

Instances	ACO-SSP (vertex)	ACO-SSP (clique)	Tabu	VNS-CSP
frb30-15	5 (**0.4**)	5 (**1.3**)	5 (**0.5**)	5 (**0.18**)
frb35-17	5 (**3.0**)	5 (**5.0**)	5 (**0.9**)	5 (**0.26**)
frb40-19	5 (**7.0**)	5 (**103.5**)	5 (**9.1**)	5 (**0.39**)
frb45-21	5 (**467.7**)	5 (**354.1**)	5 (**43.4**)	5 (**0.56**)
frb50-23	3 (**430.4**)	3 (**680.5**)	4 (**9.9**)	5 (**0.80**)
frb53-24	3 (**105.7**)	3 (**530.8**)	4 (**291.6**)	5 (**0.99**)
frb56-25	2 (**535.8**)	2 (**170.2**)	4 (**329.3**)	5 (**1.01**)
frb59-26	1 (**63.6**)	0 (—)	4 (**523.7**)	5 (**1.28**)

metaheuririics have a predefined set of parameters that has to be calibrated with respect to the problem at hand. This parameter tuning which becomes a tedious task as the number of parameter increases plays a significant impact on the solving progress and therefore the solution quality. What distinguishes the proposed metaheuristic from state-of-the-art techniques is the absence of parameter tuning making it highly suitable in practical scenarios.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References

[1] M. Dincbas and H. Simonis, "APACHE—a constraint based, automated stand allocation system," in *Proceedings of the Advanced Software Technology in Air Transport (ASTAIR '91)*, pp. 267–282, Royal Aeronautical Society, London, UK, 1991.

[2] J. Bellone, A. Chamard, and C. Pradelles, "PLANE -an evolutive planning system for aircraft production," in *Proceedings of the 1st International Conference on Practical Application of Prolog*, 1992.

[3] H. Simonis and P. Charlier, "COBRA—a system for train crew Scheduling," in *Proceedings of the Workshop on Constraint Programming and Large Scale Combinatorial Optimization (DIMACS '98)*, Rutgers University, New Brunswick, NJ, USA, September 1998.

[4] G. Baues, P. Kay, and P. Charlier, " Constraint based resource allocation for airline crew management," in *PRoceedings of the ATTIS*, Paris, France, April 1994.

[5] R. Amadini, I. Sefrioui, J. Mauro, and M. Gabbrielli, "A constraint-based model for fast post-disaster emergency vehicle routing," *International Jorunal of Interactive Multimedia and Artificial Intelligence*, vol. 2, no. 4, pp. 67–75, 2013.

[6] K. Kinoshita, K. Iizuka, and Y. Iizuka, "Effective disaster evacuation by solving the distributed constraint optimization problem," in *Proceedings of the 2nd IIAI International Conference on Advanced Applied Informatics (IIAIAAI '13)*, pp. 399–400, Los Alamitos, Calif, USA, September 2013.

[7] F. Rossi, P. V. Beek, and T. Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science, New York, NY, USA, 2006.

[8] R. Dechter and J. Pearl, "Tree clustering for constraint networks," *Artificial Intelligence*, vol. 38, no. 3, pp. 353–366, 1989.

[9] F. Rossi, C. Petri, and V. Dhar, "On the equivalence of constraint satisfaction problems," in *Proceedings of the European Conference on Artificial Intelligence (ECAI '90)*, pp. 550–556, 1990.

[10] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[11] R. Dechter and D. Frost, "Backjump-based backtracking for constraint satisfaction problems," *Artificial Intelligence*, vol. 136, no. 2, pp. 147–188, 2002.

[12] N. Jussien, G. Rochart, and X. Lorca, "Choco: an open source java constraint programming Library," in *Proceedings of the Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP '08)*, pp. 1–10, Paris, France, June 2008.

[13] S. Merchez, C. Lecoutre, and F. Boussemart, "AbsCon: a prototype to solve CSPs with abstraction," in *Principles and Practice of Constraint Programming—CP 2001*, vol. 2239 of *Lecture Notes in Computer Science*, pp. 730–744, Springer, Berlin, Germany, 2001.

[14] J.-C. Régin, "AC-*: a configurable, generic and adaptive arc consistency algorithm," in *Principles and Practice of Constraint Programming—CP 2005*, vol. 3709 of *Lecture Notes in Computer Science*, pp. 505–519, Springer, Berlin, Germany, 2005.

[15] D. Sabin and E. C. Freuder, "Contradicting conventional wisdom in constraint satisfaction," in *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI '94)*, pp. 125–129, Amsterdam, The Netherlands, August 1994.

[16] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear CSP into SAT," *Constraints*, vol. 14, no. 2, pp. 254–272, 2009.

[17] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 161–205, 1992.

[18] P. Morris, "The breakout method for escaping from local minima," in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI '93)*, pp. 40–45, 1993.

[19] R. J. Wallace and E. C. Freuder, "Heuristic methods for over-constrained constraint satisfaction problems," in *Over-Constrained Systems*, vol. 1106 of *Lecture Notes in Computer Science*, pp. 207–216, Springer, Berlin, Germany, 1996.

[20] P. Galinier and J.-K. Hao, "Tabu search for maximal constraint satisfaction problems," in *Principles and Practice of Constraint Programming-CP97*, vol. 1330 of *Lecture Notes in Computer Science*, pp. 196–208, Springer, Berlin, Germany, 1997.

[21] T. Stützle, *Local search algorithms for combinatorial problems—analysis, improvements, and new applications [Ph.D. thesis]*, TU Darmstadt, FB Informatik, Darmstadt, Germany, 1998.

[22] N. Bacanin and M. Tuba, "Artificial bee colony (ABC) algorithm for constrained optimization improved with genetic operators," *Studies in Informatics and Control*, vol. 21, no. 2, pp. 137–146, 2012.

[23] M. R. Bonyadi, X. Li, and Z. Michalewicz, "A hybrid particle swarm with velocity mutation for constraint optimization problems," in *Proceedings of the 15th Genetic and Evolutionary Computation Conference (GECCO '13)*, pp. 1–8, ACM, New York, NY, USA, July 2013.

[24] D. Curran, E. Freuder, and T. Jansen, "Incremental evolution of local search heuristics," in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO '10)*, pp. 981–982, ACM, New York, NY, USA, July 2010.

[25] S. Voß, "Meta-heuristics: state of the art," in *Local Search for Planning and Scheduling*, vol. 2148 of *Lecture Notes in Computer Science*, pp. 1–23, Springer, Berlin, Germany, 2001.

[26] Y. Zhou, G. Zhou, and J. Zhang, "A hybrid glowworm swarm optimization algorithm for constrained engineering design problems," *Applied Mathematics and Information Sciences*, vol. 7, no. 1, pp. 379–388, 2013.

[27] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu, "Genet: a connectionist architecture for solving constraint satisfaction problems by iterative improvement," in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94)*, pp. 325–330, Seattle, Wash, USA, August 1994.

[28] C. Voudouris and E. P. K. Tsang, "Guided local search," in *Handbook of Metaheuristics*, vol. 57 of *International Series in Operation Research and Management Science*, pp. 185–218, Kluwer Academic Publishers, Boston, Mass, USA, 2003.

[29] Y. Shang and B. W. Wah, "A discrete Lagrangian-based global-search method for solving satisfiability problems," *Journal of Global Optimization*, vol. 12, no. 1, pp. 61–99, 1998.

[30] D. Schuurmans, F. Southey, and R. C. Holte, "The exponentiated subgradient algorithm for heuristic Boolean programming," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI '01)*, pp. 334–341, Morgan Kaufmann, San Francisco, Calif, USA, August 2001.

[31] F. Hutter, D. A. D. Tompkins, and H. H. Hoos, "Scaling and probabilistic smoothing: efficient dynamic local search for SAT," in *Principles and Practice of Constraint Programming—CP 2002*, vol. 2470 of *Lecture Notes in Computer Science*, pp. 233–248, Springer, Berlin, Germany, 2002.

[32] D. A. H. Amante and H. T. Marin, "Adaptive penalty weights when solving congress timetabling," in *Advances in Artificial Intelligence—IBERAMIA 2004*, vol. 3315 of *Lecture Notes in Computer Science*, pp. 144–153, Springer, Berlin, Germany, 2004.

[33] M. R. Karim, "A new approach to constraint weight learning for variable ordering in CSPs," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '14)*, pp. 2716–2723, Beijing, China, July 2014.

[34] R. Shalom, M. Avigal, and R. Unger, "A conflict based SAW method for constraint satisfaction problems," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 373–380, IEEE, Trondheim, Norway, May 2009.

[35] W. Pullan, F. Mascia, and M. Brunato, "Cooperating local search for the maximum clique problem," *Journal of Heuristics*, vol. 17, no. 2, pp. 181–199, 2011.

[36] S. Fang, Y. Chu, K. Qiao, X. Feng, and K. Xu, "Combining edge weight and vertex weight for minimum vertex cover problem," in *Frontiers in Algorithmics: 8th International Workshop, FAW 2014, Zhangjiajie, China, June 28–30, 2014. Proceedings*, vol. 8497 of *Lecture Notes in Computer Science*, pp. 71–81, Springer, 2014.

[37] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI '04)*, pp. 146–150, August 2004.

[38] M.-J. Huguet, P. Lopez, and W. Karoui, "Weight-based heuristics for constraint satisfaction and combinatorial optimization problems," *Journal of Mathematical Modelling and Algorithms*, vol. 11, no. 2, pp. 193–215, 2012.

[39] M. Mouhoub and B. Jafari, "Heuristic techniques for variable and value ordering in CSPs," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pp. 457–464, ACM, Dublin, Ireland, July 2011.

[40] A. Alexiadis and J. Refanidis, "Post-optimizing individual activity plans through local search," in *Proceedings of the 8th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS '13)*, pp. 7–15, 2013.

[41] D. Joslin and D. P. Clements, "Squeaky wheel optimization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 353–373, 1999.

[42] H.-J. Lee, S.-J. Cha, Y.-H. Yu, and G.-S. Jo, "Large neighborhood search using constraint satisfaction techniques in vehicle routing problem," in *Advances in Artificial Intelligence*, vol. 5549 of *Lecture Notes in Computer Science*, pp. 229–232, Springer, Berlin, Germany, 2009.

[43] W. S. Havens and B. N. Dilkina, "A hybrid schema for systematic local search," in *Advances in Artificial Intelligence: 17th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2004, London, Ontario, Canada, May 17–19, 2004. Proceedings*, vol. 3060 of *Lecture Notes in Computer Science*, pp. 248–260, Springer, Berlin, Germany, 2004.

[44] N. Jussien and O. Lhomme, "Local search with constraint propagation and conflict-based heuristics," *Artificial Intelligence*, vol. 139, no. 1, pp. 21–45, 2002.

[45] P. V. Hentenryck and L. Michel, *Constraint-Based Local Search*, MIT Press, 2005.

[46] B. W. Kerninghan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[47] S. Lin and B. W. Kerninghan, "An effective heuristic for the traveling salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[48] U. Grenander, *Pattern Analysis*, Springer, Berlin, Germany, 1978.

[49] J. Bentley, "Programming pearls: perspective on performance," *Communications of the ACM*, vol. 27, no. 11, pp. 1087–1092, 1984.

[50] C. Lecoutre, 2010, https://www.cril.univ-artois.fr/~lecoutre/benchmarks.html.

[51] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, "Random constraint satisfaction: easy generation of hard (satisfiable) instances," *Artificial Intelligence*, vol. 171, no. 8-9, pp. 514–534, 2007.

[52] K. Xu and W. Li, "Exact phase Transition in constraint satisfaction problems," *Journal of Artificial Intelligence Research*, vol. 12, pp. 93–103, 2000.

[53] C. Solnon, *Ant Colony Optimization and Constraint Programming*, Wiley-ISTE, 2006.