Chapter 13

A Multilevel Genetic Algorithm for the Maximum Satisfaction Problem

Noureddine Bouhmala

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.78299

Abstract

Genetic algorithms (GA) which belongs to the class of evolutionary algorithms are regarded as highly successful algorithms when applied to a broad range of discrete as well continuous optimization problems. This chapter introduces a hybrid approach combining genetic algorithm with the multilevel paradigm for solving the maximum constraint satisfaction problem (Max-CSP). The multilevel paradigm refers to the process of dividing large and complex problems into smaller ones, which are hopefully much easier to solve, and then work backward toward the solution of the original problem, using the solution reached from a child level as a starting solution for the parent level. The promising performances achieved by the proposed approach are demonstrated by comparisons made to solve conventional random benchmark problems.

Keywords: maximum constraint satisfaction problem, genetic algorithms, multilevel paradigm

1. Introduction

Many problems in the field of artificial intelligence can be modeled as constraint satisfaction problems (CSP). A CSP is a tuple $\langle X, D, C\rangle$ where, $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $D = \{D_{x_1}, D_{x_2}, \ldots, D_{x_n}\}$ is a finite set of domains. Thus each variable $x \in X$ has a corresponding discrete domain D_x from which it can be instantiated, and $C = \{C_1, C_2, \ldots, C_k\}$ 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 $D_1 \times \ldots \times D_k$, each element of which are values that the variables cannot take simultaneously. 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. The maximum constraint satisfaction problem (Max-CSP) aims at finding an assignment so as to maximize the number of

satisfied constraints. Max-CSP can be regarded as the generalization of CSP; the solution maximizes the number of satisfied constraints. In this chapter, 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 non-binary CSP can theoretically be converted to a binary CSP [1]. Algorithms for solving CSPs apply 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. Examples include the minimum conflict heuristic MCH [2], the break method for escaping from local minima [3], and various enhanced MCH (e.g., randomized iterative improvement of MCH called WMCH [4], MCH with tabu search [5], and evolutionary algorithms [6]). Algorithms based on assigning weights on constraints are techniques that work by introducing weights on variables or constraints in order to avoid local minima. Methods belonging to this category include genet [7], guided local search [8], the exponentiated subgradient [9], discrete Lagrangian search [10], the scaling and probabilistic smoothing [11], evolutionary algorithms combined with stepwise adaptation of weights [12], methods based on dynamically adapting weights on variables [13], or both (i.e., variables and constraints) [14]. Methods based on large neighborhood search have recently attracted several researchers for solving the CSP [15]. The central idea is to reduce the size of local search space relying on a continual relaxation (removing elements from the solution) and re-optimization (re-inserting the removed elements). Finally, the work introduced in [16] introduces a variable depth metaheuristic combing a greedy local search with a self-adaptive weighting strategy on the constraints weights.

2. Algorithm

2.1. Multilevel context

The multilevel paradigm is a simple technique, which at its core involves recursive coarsening to produce smaller and smaller problems that are easier to solve than the original one. Multilevel techniques have been developed in the period after 1960 and are among the most efficient techniques used for solving large algebraic systems arising from the discretization of partial differential equations. In recent years, it has been recognized that an effective way of enhancing metaheuristics is to use them in the multilevel context. The pseudo-code of the multilevel genetic algorithm is shown in **Algorithm 1**. **Figure 1** illustrates the multilevel paradigm used for six variables and two coarsening levels. The multilevel paradigm consists of four phases: coarsening, initial solution, uncoarsening, and refinement. The coarsening phase aims at merging the variables associated with the problem to form clusters. The clusters are used in a recursive manner to construct a hierarchy of problems each representing the original problem but with fewer degrees of freedom. The coarsest level can then be used to compute an initial solution. The solution found at the coarsest level is uncoarsened (extended to give an initial solution for the parent level) and then improved using a chosen optimization algorithm. A common feature that characterizes multilevel algorithms, is that any solution in any of the coarsened problems is a legitimate solution to the original one. Optimization algorithms using the multilevel paradigm draw their strength from coupling the refinement process across different levels.

http://dx.doi.org/10.5772/intechopen.78299

input : Problem P_0
output: Solution $S_{final}(P_0)$
begin
  level := 0;
  while *Not reached the desired number of levels* **do**
    $P_{level+1}$:= Coarsen (P_{level});
    level := level + 1;
  end
  /* Initial Solution is computed at the lowest level */;
  $S(P_{level})$ =Initial Solution (P_{level}) ;
  while ($level > 0$) **do**
    $S_{start}(P_{level-1})$: = Uncoarsen ($S_{final}(P_{level})$);
    $S_{final}(P_{level-1})$:= Refine ($S_{start}(P_{level-1})$);
    level := level - 1;
  end
end

Algorithm 1. The multilevel genetic algorithm.

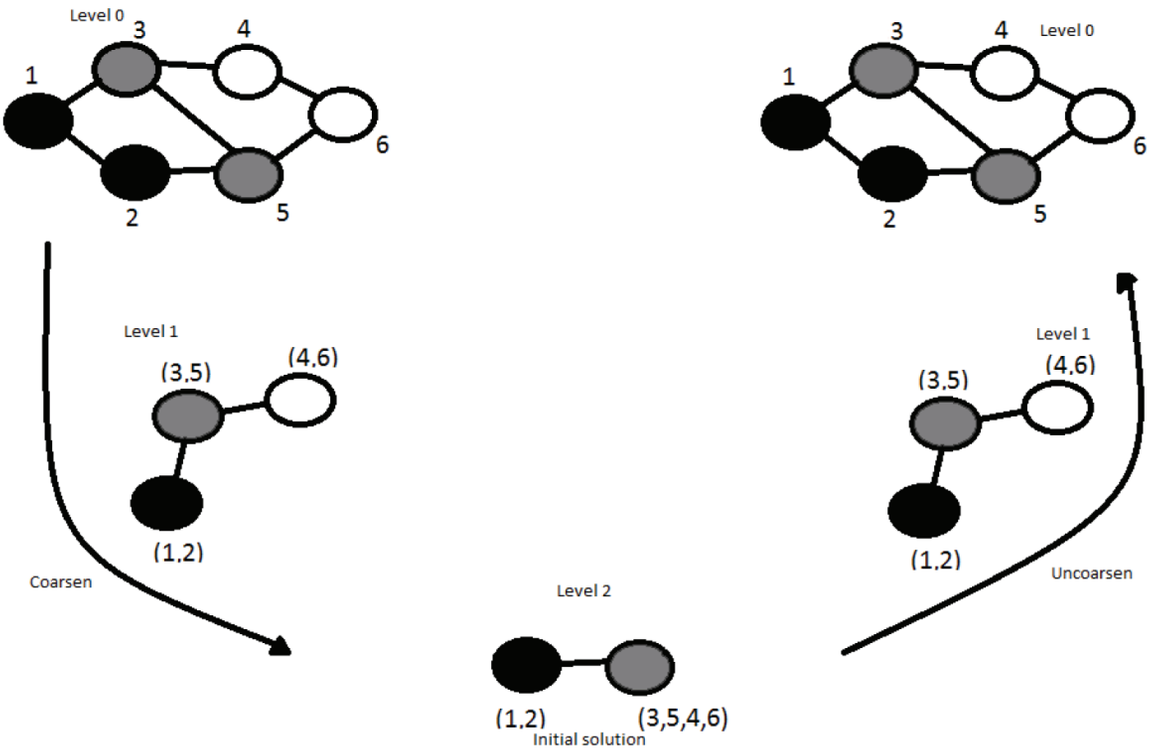

Figure 1. The different steps of the multilevel paradigm.

2.2. Multilevel genetic algorithm (GA)

GAs [17] are stochastic methods for global search and optimization and belong to the group of nature-inspired metaheuristics leading to the so-called natural computing. It is a fast-growing interdisciplinary field in which a range of techniques and methods are studied for dealing with large, complex, and dynamic problems with various sources of potential uncertainties. GAs simultaneously examine and manipulate a set of possible solutions. A gene is a part of a chromosome (solution), which is the smallest unit of genetic information. Every gene is able to assume different values called allele. All genes of an organism form a genome, which affects the appearance of an organism called phenotype. The chromosomes are encoded using a chosen representation and each can be thought of as a point in the search space of candidate solutions. Each individual is assigned a score (fitness) value that allows assessing its quality. The members of the initial population may be randomly generated or by using sophisticated mechanisms by means of which an initial population of high-quality chromosomes is produced. The reproduction operator selects (randomly or based on the individual's fitness) chromosomes from the population to be parents and enter them in a mating pool. Parent individuals are drawn from the mating pool and combined so that information is exchanged and passed to off-springs depending on the probability of the crossover operator. The new population is then subjected to mutation and enters into an intermediate population. The mutation operator acts as an element of diversity into the population and is generally applied with a low-probability to avoid disrupting crossover results. Finally, a selection scheme is used to update the population giving rise to a new generation. The individuals from the set of solutions, which is called population will evolve from generation to generation by repeated applications of an evaluation procedure that is based on genetic operators. Over many generations, the population becomes increasingly uniform until it ultimately converges to optimal or near-optimal solutions. The different steps of the multilevel weighted genetic algorithm are described as follows:

- *construction of levels*: let $G_0 = (V_0, E_0)$ be an undirected graph of vertices V and edges E. The set V denotes variables and each edge $(x_i, x_j) \in E$ implies a constraint joining the variables x_i and x_j. Given the initial graph G_0, the graph is repeatedly transformed into smaller and smaller graphs G_1, G_2, …, G_m such that $|V_0| > |V_1| > \ldots > |V_m|$. To coarsen a graph from G_j to G_{j+1}, a number of different techniques may be used. In this chapter, when combining a set of variables into clusters, the variables are visited in a random order. If a variable x_i has not been matched yet, then the algorithms randomly select one of its neighboring unmatched variable x_j, and a new cluster consisting of these two variables is created. Its neighbors are the combined neighbors of the merged variables x_i and x_j. Unmatched variables are simply left unmatched and copied to the next level.

- *initial assignment*: the process of constructing a hierarchy of graphs ceases as soon as the size of the coarsest graphs reaches some desired threshold. A random initial population is generated at the lowest level $G_k = (V_k, E_k)$. The chromosomes, which are assignments of values to the variables are encoded as strings of bits, the length of which is the number of variables. At the lowest level, the length of the chromosome is equal to the number of clusters. The initial solution is simply constructed by assigning to all variable in a cluster, a random value v_i. In this work, it is assumed that all variables have the same domain

http://dx.doi.org/10.5772/intechopen.78299

(i.e., same set of values), otherwise different random values should be assigned to each variable in the cluster. All the individuals of the initial population are evaluated and assigned a fitness expressed in Eq. (1), which counts the number of constraint violations where $< (x_i, s_i), (x_j, s_j) >$ denotes the constraint between the variables x_i and x_j where x_i is assigned the value s_i from D_{x_i} and x_j is assigned the value s_j from D_{x_j}.

$$Fitness = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Violation\left(W_{i,j} < (x_i, s_i), (x_j, s_j) >\right) \quad (1)$$

- *initial weights*: the next step of the algorithm assigns a fixed amount of weight equal to 1 across all the constraints. The distribution of weights to constraints aims at forcing hard constraints with large weights to be satisfied thereby preventing the algorithm at a later stage from getting stuck at a local optimum.
- *optimization*: having computed an initial solution at the coarsest graph, GA starts the search process from the coarsest level $G_k = (V_k, E_k)$ and continues to move toward smaller levels. The motivation behind this strategy is that the order in which the levels are traversed offers a better mechanism for performing diversification and intensification. The coarsest level allows GA to view any cluster of variables as a single entity leading the search to become guided in faraway regions of the solution space and restricted to only those configurations in the solution space in which the variables grouped within a cluster are assigned the same value. As the switch from one level to another implies a decrease in the size of the neighborhood, the search is intensified around solutions from previous levels in order to reach better ones.
- *parent selection*: during the optimization, new solutions are created by combining pairs of individuals in the population and then applying a crossover operator to each chosen pair. Combining pairs of individuals can be viewed as a matching process. In the version of GA used in this work, the individuals are visited in random order. An unmatched individual i_k is matched randomly with an unmatched individual i_l.
- *genetic operators*: the task of the crossover operator is to reach regions of the search space with higher average quality. The two-point crossover operator is applied to each matched pair of individuals. The two-point crossover selects two randomly points within a chromosome and then interchanges the two parent chromosomes between these points to generate two new offspring.
- *survivor selection*: the selection acts on individuals in the current population. Based on each individual quality (fitness), it determines the next population. In the roulette method, the selection is stochastic and biased toward the best individuals. The first step is to calculate the cumulative fitness of the whole population through the sum of the fitness of all individuals. After that, the probability of selection is calculated for each individual as being $P_{Selection_i} = f_i / \sum_1^N f_i$, where f_i is the fitness of individual i.
- *updating weights*: the weights of each current violated constraint is then increased by one, whereas the newly satisfied constraints will have their weights decreased by one before the start of new generation.

- *termination condition*: the convergence of GA is supposed to be reached if the best individual remains unchanged during five consecutive generations.
- *projection*: once GA has reached the convergence criterion with respect to a child level graph $G_k = (V_k, E_k)$, the assignment reached on that level must be projected on its parent graph $G_{k-1} = (V_{k-1}, E_{k-1})$. The projection algorithm is simple; if a cluster belongs to $G_k = (V_k, E_k)$ is assigned the value vl_i, the merged pair of clusters that it represents belonging to $G_{k-1} = (V_{k-1}, E_{k-1})$ are also assigned the value vl_i,

3. Experimental results

3.1. Experimental setup

The benchmark instances were generated using model A [18] as follows: each instance is defined by the 4-tuple n, m, p_d, p_t, where n is the number of variables; m is the size of each variable's domain; p_d, the constraint density, is the proportion of pairs of variables, which have a constraint between them; and p_t, the constraint tightness, is the probability that a pair of values is inconsistent. From the $(n \times (n-1)/2)$ possible constraints, each one is independently chosen to be added in the constraint graph with the probability p_d. Given a constraint, we select with the probability p_t, which value pairs become no-goods. The model A will on average have $p_d \times (n-1)/2$ constraints, each of which has on average $p_t \times m^2$ inconsistent pairs of values. For each pair of density tightness, we generate one soluble instance (i.e., at least one solution exists). Because of the stochastic nature of GA, we let each algorithm do 100 independent runs, each run with a different random seed. Many NP-complete or NP-hard problems show a phase transition point that marks the spot where we go from problems that are under-constrained and so relatively easy to solve, to problems that are over-constrained and so relatively easy to prove insoluble. Problems that are on average harder to solve occur between these two types of relatively easy problem. The values of p_d and p_t are chosen in such a way that the instances generated are within the phase transition. In order to predict the phase transition region, a formula for the constrainedness [19] of binary CSPs was defined by:

$$\kappa = \frac{n-1}{2} p_d \, log_m \left(\frac{1}{1-pt} \right). \tag{2}$$

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. The following parameters have been fixed experimentally and are listed below:

- Population size = 50
- Stopping criteria for the coarsening phase: the reduction process stops as soon as the number of levels reaches 3. At this level, MLV-WGA generates an initial population.
- Convergence during the optimization phase: if there is no observable improvement of the fitness function of the best individual during five consecutive generations, MLV-WGA is assumed to have reached convergence and moves to a higher level.

http://dx.doi.org/10.5772/intechopen.78299

3.2. Results

The plots in **Figures 2** and **3** compare the WGA with its multilevel variant MLV-WGA. The improvement in quality imparted by the multilevel context is immediately clear. Both WGA and MLV-WGA exhibit what is called a plateau region. A plateau region spans a region in the search space where crossover and mutation operators leave the best solution or the mean solution unchanged. However, the length of this region is shorter with MLV-WGA compared to that of WGA. The multilevel context uses the projected solution obtained at $G_{m+1}(V_{m+1}, E_{m+1})$ as the initial solution for $G_m(V_m, E_m)$ for further refinement. Even though the solution at $G_{m+1}(V_{m+1}, E_{m+1})$ is at a local minimum, the projected solution may not be at a local optimum with respect to $G_m(V_m, E_m)$. The projected assignment is already a good solution leading WGA to converge quicker within few generations to a better solution. **Tables 1–3** show a comparison of

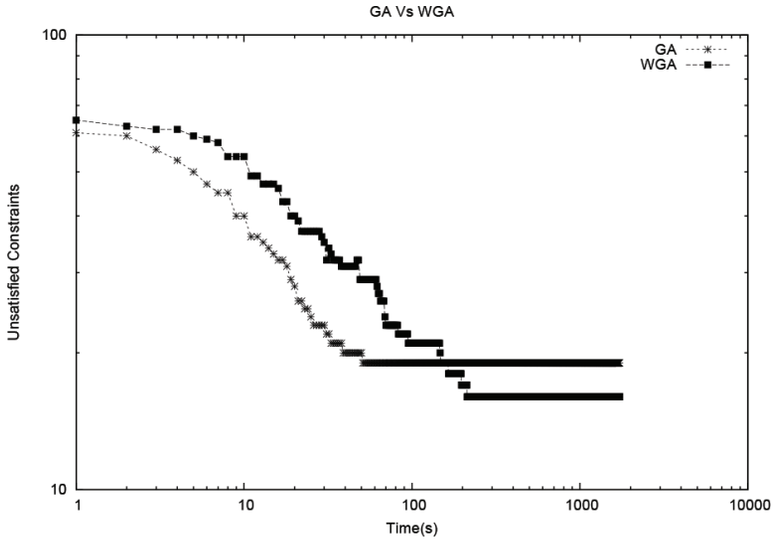

Figure 2. MLV-GA vs. GA: evolution of the mean unsatisfied constraints as a function of time. Csp-N30-DS40-C125-cd026ct063.

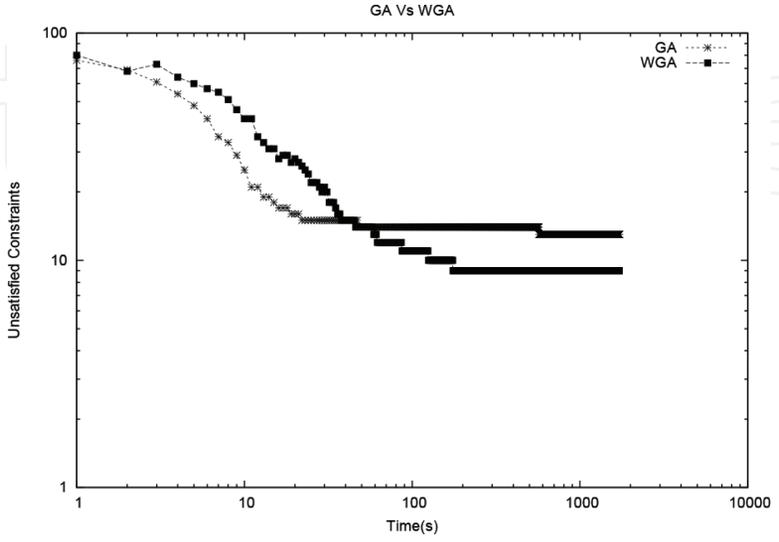

Figure 3. MLV-GA vs. GA: evolution of the mean unsatisfied constraints as a function of time. Csp-N35-DS20-C562-cd094-ct017.

	MLV-WGA				WGA			
Instance	**Min**	**Max**	**Mean**	RE_{av}	**Min**	**Max**	**Mean**	RE_{av}
N25-DS20-C36-cd-014-ct083	3	7	4.58	**0.128**	3	8	5.41	0.151
N25-DS20-C44-cd012-ct087	6	10	8.04	**0.183**	8	14	9.91	0.226
N25-DS20-C54-cd018-ct075	3	7	5.37	**0.100**	4	9	6.91	0.128
N25-DS20-C78-cd026-ct061	2	8	4.33	**0.056**	2	10	5.79	0.073
N25-DS20-C225-cd078-ct027	3	8	4.16	**0.019**	3	9	5.66	0.026
N25-DS20-C229-cd072-ct029	4	9	6.04	**0.014**	4	11	8.16	0.036
N25-DS20-C242-cd086-ct025	1	6	3.5	**0.015**	3	10	5.70	0.024
N25-DS20-C269-cd086-ct025	4	10	5.66	**0.022**	4	10	7.54	0.029
N25-DS20-C279-cd094-ct023	2	7	4.75	**0.018**	4	9	6.75	0.025
N25-DS40-C53-cd016-ct085	6	11	8.91	**0.169**	8	13	10.70	0.202
N25-DS40-C70-cd026-ct069	2	6	4.25	**0.061**	3	8	5.75	0.083
N25-DS40-C72-cd022-ct075	6	12	9	**0.125**	6	15	10.45	0.146
N25-DS40-C102-cd032-ct061	5	12	8.12	**0.080**	7	14	10.33	0.102
N25-DS40-C103-cd034-ct059	5	9	6.83	**0.067**	4	12	8.79	0.086
N25-DS40-C237-cd082-ct031	3	8	5.66	**0.024**	5	10	7.87	0.034
N25-DS40-C253-cd088-ct029	3	7	4.95	**0.020**	5	12	8.04	0.032
N25-DS40-C264-cd088-ct029	5	10	6.91	**0.027**	6	16	8.91	0.034
N25-DS40-C281-cd096-ct027	3	9	5.62	**0.020**	4	12	8.54	0.031
N25-DS40-C290-cd096-ct027	4	10	7.08	**0.025**	6	14	9	0.032

REav denotes the relative error in percent. The value in bold shows the algorithm with the lowest RE.

Table 1. MLV-WGA vs. WGA: number of variables: 25.

	MLV-WGA				WGA			
Instance	**Min**	**Max**	**Mean**	RE_{av}	**Min**	**Max**	**Mean**	RE_{av}
N30-DS20-C41-cd012-ct083	2	6	3.70	**0.026**	3	7	5.08	0.124
N30-DS20-C71-cd018-ct069	1	7	3.37	**0.048**	3	10	5.66	0.080
N30-DS20-C85-cd020-ct065	3	9	6	**0.071**	5	12	8.37	0.099
N30-DS20-C119-cd028-ct053	3	10	5.70	**0.048**	6	12	8.83	0.075
N30-DS20-C334-cd074-ct025	6	13	8.16	**0.025**	6	14	9.87	0.030
N30-DS20-C387-cd090-ct021	3	9	6.66	**0.018**	5	13	8.70	0.033
N30-DS20-C389-cd090-ct021	2	9	6.08	**0.016**	4	14	8.95	0.024
N30-DS20-C392-cd090-ct021	3	10	7.08	**0.019**	5	15	9.16	0.024
N30-DS20-C399-cd090-ct021	5	13	7.70	**0.020**	6	14	9.79	0.025
N30-DS40-C85-cd020-ct073	5	11	7.75	**0.092**	7	14	10.87	0.152
N30-DS40-C96-cd020-ct073	8	12	16	0.167	11	19	14.58	0.015

	MLV-WGA				WGA			
Instance	**Min**	**Max**	**Mean**	RE_{av}	**Min**	**Max**	**Mean**	RE_{av}
N30-DS40-C121-cd026-ct063	8	14	10.5	**0.087**	9	19	14.33	0.152
N30-DS40-C125-cd026-ct063	8	18	12.20	**0.098**	10	19	15.58	0.125
N30-DS40-C173-cd044-ct045	4	10	6.41	**0.038**	6	14	9.20	0.054
N30-DS40-C312-cd070-ct031	7	14	10.5	0.033	7	19	13.33	0.025
N30-DS40-C328-cd076-ct029	6	13	10.37	**0.032**	10	18	13.45	0.042
N30-DS40-C333-cd076-ct029	7	13	10.25	**0.031**	9	18	12.62	0.038
N30-DS40-C389-cd090-ct025	6	13	9.33	**0.024**	9	17	12.20	0.032
N30-DS40-C390-cd090-ct025	6	14	9.29	**0.024**	10	17	13	0.031

REav denotes the relative error in percent. The value in bold shows the algorithm with the lowest RE.

Table 2. MLV-WGA vs. WGA: number of variables: 30.

	MLV-WGA				WGA			
Instance	**Min**	**Max**	**Mean**	RE_{av}	**Min**	**Max**	**Mean**	RE_{av}
N40-DS20-C78-cd010-ct079	6	12	8.91	**0.115**	5	13	9.04	0.116
N40-DS20-C80-cd010-ct079	7	13	9.62	**0.121**	7	13	10.04	0.153
N40-DS20-C82-cd012-ct073	4	9	6.25	**0.073**	4	11	6.95	0.085
N40-DS20-C95-cd014-ct067	2	8	4.45	0.047	2	7	4.12	**0.044**
N40-DS20-C653-cd084-ct017	2	14	9.37	**0.015**	**6**	**16**	**10.62**	**0.018**
N40-DS20-C660-cd084-ct017	6	14	9.12	**0.014**	7	6	9.75	0.015
N40-DS20-C751-cd096-ct015	6	13	9.91	0.014	5	13	9.83	0.014
N40-DS20-C752-cd096-ct015	5	17	9.29	0.013	3	13	9.20	0.013
N40-DS20-C756-cd096-ct015	6	15	9.95	0.014	5	16	8.75	**0.012**
N40-DS40-C106-cd014-ct075	7	14	11.08	**0.105**	7	16	11.5	0.109
N40-DS40-C115-cd014-ct075	12	20	15.5	0.135	11	20	15.5	0.135
N40-DS40-C181-cd024-ct055	6	17	12.04	0.067	7	17	11.75	**0.065**
N40-DS40-C196-cd024-ct055	11	12	16.58	0.085	12	20	15.54	**0.080**
N40-DS40-C226-cd030-ct047	7	14	10.91	0.051	7	16	11.16	**0.050**
N40-DS40-C647-cd082-ct021	11	23	15.66	0.025	11	20	15.20	**0.024**
N40-DS40-C658-cd082-ct021	11	22	16.33	**0.025**	13	21	16.70	0.026
N40-DS40-C703-cd092-ct019	9	21	13.41	0.020	8	20	13.58	0.020
N40-DS40-C711-cd092-ct019	12	23	15.75	0.023	8	20	14.87	0.021
N40-DS40-C719-cd092-ct019	8	21	16.54	0.024	10	20	15.16	**0.022**

REav denotes the relative error in percent. The value in bold shows the algorithm with the lowest RE.

Table 3. MLV-WGA vs. WGA: number of variables 40.

the two algorithms. For each algorithm, the best (Min) and the worst (Max) results are given, while mean represents the average solution. MLV-WGA outperforms WGA in 53 cases out of 96, gives similar results in 20 cases, and was beaten in 23 cases. The performance of both algorithms differs significantly. The difference for the total performance is between 25 and 70% in the advantage of MLV-GA. Comparing the worst performances of both algorithms, MLV-WGA gave bad results in 15 cases, both algorithms give similar results in 8 cases, and MLV-WGA was able to perform better than WGA in 73 cases. Looking at the average results, MLV-WGA does between 16 and 41% better than WGA in 84 cases, while the differences are very marginal in the remaining cases where WGA beats MLV-WGA.

4. Conclusion

In this work, a multilevel weighted based-genetic algorithm is introduced for MAX-CSP. The results have shown that the multilevel genetic algorithm returns a better solution for the equivalent run-time for most cases compared to the standard genetic algorithm. The multilevel paradigm offeres a better strategy for performing diversification and intensification. This is achieved by allowing GA to view a cluster of variables as a single entity thereby leading the search becoming guided and restricted to only those assignments in the solution space in which the variables grouped within a cluster are assigned the same value. As the size of the clusters gets smaller from one level to another, the size of the neighborhood becomes adaptive, and allows the possibility of exploring different regions in the search space while intensifying the search by exploiting the solutions from previous levels in order to reach better solutions.

Author details

Noureddine Bouhmala

Address all correspondence to: noureddine.bouhmala@usn.no

Department of Maritime Technology and Innovation, University SouthEast, Raveien, Borre, Norway

References

[1] Dechter R, Pearl J. Tree clustering for constraint networks. Artificial Intelligence. 1989;**38**: 353366

[2] Minton S, Johnson M, Philips A, Laird P. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling scheduling problems. Artificial Intelligence. 1992;**58**:161-205

[3] Morris P. The breakout method for escaping from local minima. In: Proceeding AAAI'93 Proceedings of the Eleventh National Conference on Artificial Intelligence. 1993. pp. 40-45

[4] Wallace R, Freuder E. Heuristic methods for over-constrained constraint satisfaction problems. In: Over-Constrained Systems. LNCS. Vol. 1106. Berlin, Germany: Springer Verlag; 1995. pp. 207-216

[5] Galinier P, Hao, J. Tabu search for maximal constraint satisfaction problems. In: Principles and Practice of Constraint Programming CP 1997. LNCS. Vol. 1330. Berlin, Germany: Springer Verlag; 1997. pp. 196-208

[6] Zhou Y, Zhou G, Zhang J. A hybrid glowworm swarm optimization algorithm for constrained engineering design problems. Applied Mathematics and Information Sciences. 2013;**7**(1):379-388

[7] Davenport A, Tsang E, Wang C, Zhu K. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In: Proceedings of the Twelth National Conference on Artificial Intelligence. 1994

[8] Voudouris C, Tsang E. Guided local search: Handbook of metaheuristics. International Series in Operation Research and Management Science. 2003;**57**:185-218

[9] Schuurmans D, Southey F, Holte E. The exponentiated subgradient algorithm for heuristic Boolean programming. In: 17th International Joint Conference on Artificial Intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers; 2001. pp. 334-341

[10] Shang E, Wah B. A discrete Lagrangian-based global-search method for solving satisfiability problems. Journal of Global Optimization. 1998;**12**(1):6199

[11] Hutter F, Tompkins D, Hoos H. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Principles and Practice of Constraint Programming CP 2002. LNCS. Vol. 2470. Berlin, Germany: Springer Verlag; 2002. pp. 233-248

[12] Amante D, Marin A. Adaptive penalty weights when solving congress timetabling. Advances in Artificial Intelligence, Lectures Notes in Computer Science. 2004;**3315**:144-153

[13] Pullan W, Mascia F, Brunato M. Cooperating local search for the maximum clique problems. Journal of Heuristics. 2011;**17**:181-199

[14] Fang S, Chu Y, Qiao K, Feng X, Xu K. Combining edge weight and vertex weight for minimum vertex cover problem. In: FAW 2014. 2014. pp. 71-81

[15] Lee H, Cha S, Yu Y, Jo G. Large neighborhood search using constraint satisfaction techniques in vehicle routing problem. In: Gao Y, Japkowicz N, editors. Advances in Artificial Intelligence. Lecture Notes in Computer Science. Vol. 5549. Heidelberg: Springer Berlin; 2010. pp. 229-232

[16] Bouhmala N. A variable depth search algorithm for binary constraint satisfaction problems. Mathematical Problems in Engineering. 2015;**2015**:Article ID 637809, 10 pages. DOI: 10.1155/2015/637809

[17] Holland J. Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press; 1975

[18] Xu W. Satisfiability transition and experiments on a random constraint satisfaction problem model. International Journal of Hybrid Information Technology. 2014;7(2):191-202

[19] Gent IP, MacIntyre E, Prosser P, Walsh T. The constrainedness of search. In: Proceedings of the AAAI-96. 1996. pp. 246-252

IntechOpen