Combinatorial Hill Climbing Using Micro-Genetic Algorithms

Spyros A. Kazarlis
Technological Educational Institute of Serres,
Dept. Of Informatics & Communications
Terma Magnesias St., 621 24, Serres, GREECE
kazarlis@teiser.gr

Abstract- **This paper introduces a new hill-climbing operator, (MGAC), for GA optimization of combinatorial problems, and proposes two implementation techniques for it. The MGAC operator uses a small size second-level GA with a small population that evolves for a few generations and serves as the engine for finding better solutions in the neighborhood of the ones produced by the main GA. The two implementations are tested on a Power Systems' problem called the Unit Commitment Problem, and compared with three other methods: a GA with classic hill-climbers, Lagrangian-Relaxation, and Dynamic Programming. The results show the superiority of the proposed MGAC operator.**

I. Introduction

In order to boost the convergence speed of GAs towards the exact optimum, many hybrid genetic schemes have been proposed in the literature that combine GAs with hill climbing or local search techniques [1], [6], [8], [9], [10], [11], [12], [14], to solve both continuous variable and combinatorial problems. Memetic Algorithms [13] are probably the most well known paradigm of such schemes. Most GA-hill climbing hybrids are implemented for continuous variable problems. In such problems the hill climbers used are often designed to perform independent steps along each axis in the space (one variable at a time) searching for better solutions. One such operator is the PhenoMute (PM) hill climbing operator suggested in [2], [11], [12]. Such operators cannot follow the potential "ridges" created in the search space of difficult constrained optimization problems, as the direction of the "ridge" usually does not coincide with that of a single axis.

This problem has led the author together with other researchers in introducing the Micro GA hill climbing operator (MGA) [4], [5]. The MGA operator uses a small second-level population, or a Micro GA [3], [7], that evolves for a small number of generations and acts in a small neighborhood around the best solution produced by the main GA at each generation. The MGA operator is capable of genetically evolving paths of arbitrary direction leading to better solutions and following potential ridges in the search space regardless of their direction, width, or even discontinuities. Although proven effective the MGA operator was designed only for continuous variable problems.

In this paper the combinatorial version of the MGA operator is presented, called the Micro GA combinatorial hill climbing operator, or MGAC for brevity. The major difference in combinatorial problems is that the definition of the "neighborhood" of a solution is not very easily conceived, as every small perturbation to a combinatorial problem solution may be considered to reside within the "neighborhood" of the solution. Thus, before introducing the new MGAC operator, a definition of the combinatorial "neighborhood" is given.

Moreover, two specific implementations of MGAC are proposed, namely MGAC-ARM and MGAC-CNS. The first one (MGAC-ARM) genetically searches the space of possible perturbations within a single neighborhood at a time, but the specific neighborhood that it searches is selected among all possible neighborhoods using an Adaptive Ranking Multi-neighborhood scheme, that promotes "fertile" neighborhoods. The second one (MGAC-CNS) uses the Micro GA to search the space of possible neighborhoods (Combinatorial Neighborhood Space) and each neighborhood is evaluated by checking the quality of a number of random chosen sample perturbations within the neighborhood.

The organization of the paper is as follows: in section II the definition of the combinatorial neighborhood is given. In section III the MGAC-ARM implementation is described in detail, while the MGAC-CNS implementation is described in section IV. Section V presents the test problem on which the new operators are tested, together with the simulation results. Conclusions of this work are presented in section VI.

II. Definition of the Combinatorial Neighborhood

In order to give a definition of the combinatorial "neighborhood" that is essential to develop the MGAC operator, a few principles must be considered: a) the neighborhood must be small compared to the whole solution, b) the neighborhood must be allocated "around" the original solution, and c) perturbations within the neighborhood may result in small alterations of the whole solution. With these requirements in mind the definition of the combinatorial "neighborhood" can de formed as follows:

Without loss of generality we can assume that every combinatorial problem can be encoded using the binary alphabet. This means that every possible solution can be represented as a binary string of some length, depending on the specific problem. Lets define this length as ℓ. Then the search space SS is composed of 2^{ℓ} different solutions. The neighborhood of every solution S $\in$ SS can be defined as a subset N_S of SS, that is composed of solutions produced from

The Project is co-funded by the European Social Fund and National Resources - (EPEAEK-II) ARXIMHDHS.

solution S, by allowing n of the ℓ bits of solution S to change and keeping the rest ℓ-n bits constant. Thus, the number of solutions that reside within the neighborhood N_S is 2^n. This definition covers the requirements described earlier in this chapter as a) by keeping n small, the resulting neighborhood search space can be small compared to the original problem's search space, b) the neighborhood is around the original solution as ℓ-n bits of the solution are kept unchanged, and the rest are allowed to perturb, and c) by selecting the n bits in such a way that they are semantically close (their positions affect the same or similar regions of the decoded real solution), it can be ensured that perturbations within the neighborhood will result in alterations of certain portions of the whole solution.

From the above definition it is clear that there isn't only one single neighborhood that can be defined for every solution S. In fact the number of different neighborhoods (NN) is ℓCn, where xCy is the combination of x elements taken as groups of y elements and is given by :

$$NN = xCy = \frac{x!}{y!(x-y)!} \qquad (1)$$

In a real implementation, though, this number may be reduced when considering the third requirement (requirement c). According to this requirement, not all combinations may be considered as neighborhoods, because the selected bits of each neighborhood must be semantically close.

III. Implementation 1 (MGAC-ARM): Micro-GA searches the neighborhood selected by a Ranking Algorithm

According to this variant, the MGAC operator searches one neighborhood at a time, using a Micro GA. In other words the MGAC operator genetically scans the space of all possible perturbations of the specific n bits of a single neighborhood (n<<ℓ). The neighborhood searched by the MGAC operator should not remain the same during the main GA's run, but it should be possible for the MGAC to examine a large number of neighborhoods. Moreover, if the MGAC operator succeeds in improving the best-so-far solution by examining a specific neighborhood NEi, (i=1..NN), it is wise to insist on examining this neighborhood, as it is possible to come up with even better solutions in the future.

With the above considerations in mind, an adaptive ranked based multi-neighborhood scheme (ARM) has been developed for this MGAC variation. This scheme works as follows :

1. Before the beginning of the GA evolution, all the possible neighborhoods are calculated, and each one is assigned a neighborhood identification number and a rank. At the beginning all ranks are set equal to 1, which means that at the beginning all neighborhoods have equal probability of being selected by the MGAC operator for examination.
2. Every time the MGAC is invoked, it selects one neighborhood out of NN, with probability proportional to the neighborhood's rank (roulette wheel selection).
3. If the MGAC finds a better solution by examining the selected neighborhood, it increases its rank by 2, with a maximum of 10.
4. If it fails to find a better solution, it decreases its rank by 1, with a minimum of 1.

In the long run, neighborhoods that produce better solutions consistently, are ranked better than the ones that don't produce better solutions or the ones that managed to produce a better solution once, but proved to be unproductive later on.

The MGAC operator itself works as follows (also see Figure 1):

1. When invoked, the MGAC is fed with the best-so-far solution S_{best} of the main GA.
2. Then, it selects a neighborhood NEi with probability proportional to the neighborhoods rank, via roulette wheel selection.
3. All the bits of S_{best} that do not belong to NEi are kept unchanged. Those that belong to NEi are allowed to change.
4. It forms a population of 5 solutions that are bit strings of length n, (where n is the number of bits allowed to change), randomly generated at the beginning.
5. It evaluates each of the 5 solutions by injecting the bits of each solution to the corresponding bits of solution S_{best}.

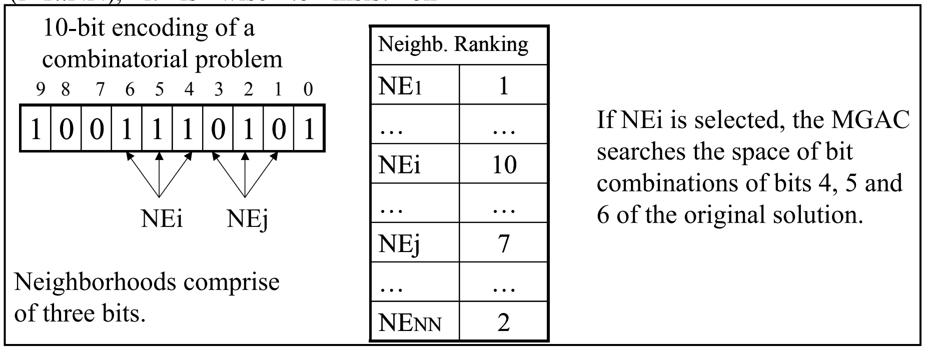

10-bit encoding of a combinatorial problem
9 8 7 6 5 4 3 2 1 0
1 0 0 1 1 1 0 1 0 1
NEi
NEj
Neighborhoods comprise of three bits.
Neighb. Ranking
NE1 1
… …
NEi 10
… …
NEj 7
… …
NENN 2
If NEi is selected, the MGAC searches the space of bit combinations of bits 4, 5 and 6 of the original solution.

Figure. 1. MGAC-ARM example on a combinatorial problem with a 10-bit encoding and neighborhoods of 3 bits. Possible neighborhoods are 10C3, i.e. 10!/(3!(7!))=120. Neighborhoods are ranked depending on whether they produce better solutions or not. Neighborhoods are selected with probability proportional to their ranks.

6. Then solutions (among the 5) are selected in pairs, with probability proportional to their fitness, to mate recombine and produce 5 new solutions (the next generation).
7. The steps 5-6 are repeated for 7 generations.
8. If the best solution produced has better fitness than the original solution S_{best}, then the rank of neighborhood NEi is increased by 2 (max 10). Otherwise, it is decreased by 1 (min 1).
9. Finally, if the best solution produced is better than S_{best}, it replaces it in the main GA's population.

From the above algorithmic description it is evident that With the MGAC-ARM scheme the following targets are achieved:

1. The neighborhood of the best-so-far solution is genetically searched by a Micro GA
2. All neighborhoods have the opportunity to be searched by the operator
3. "Fertile" neighborhoods are searched more often than "sterile" ones

However, the MGAC-ARM scheme has a scaling-up problem: when dealing with large scale combinatorial problems, the number NN of possible neighborhoods (sets of n out of ℓ bits) can be extremely large, and make it practically impossible for the algorithm to a-priori calculate and enumerate all possible neighborhoods.

IV. Implementation 2 (MGAC-CNS): Micro-GA searches the neighborhood space and each neighborhood is evaluated by samples

This scheme tries to solve the scaling-up problem of the MGAC-ARM variation. Instead of using the Micro GA to search a single neighborhood at a time (as in MGAC-ARM), that is selected among all possible neighborhoods via a ranking strategy, the MGAC-CNS variation uses the Micro GA to search the space of possible neighborhoods, in order to discover regions of "fertile" neighborhoods, the perturbations of which may give better solutions.

In order for the MGAC-CNS to work, a representation method is needed, to encode all possible neighborhoods in a string of symbols. For example, in a combinatorial problem with a 10-bit encoding and 3-bit neighborhoods, as in the previous section (Fig.1), one might adopt an integer encoding to represent possible neighborhoods. Each MGAC-CNS solution could be a vector S consisting of three (3) integers S_i, i=1..3, each of which can take values in the range 0..9, representing a bit position in the 10-bit solution of the main problem. Thus, the solution S=(4,5,6) coincides with neighborhood NEi of Figure 1. Of course care should be taken during the reproduction phase so that there are no duplicate values in every produced solution S (e.g. S=(2,2,6) is invalid).

Every time it is called, the MGAC-CNS operator initially produces a population of 5 such vectors at random. Then, it evolves this population, using common integer crossover and mutation operators, for 7 generations. Every produced vector S, that represents a specific 3-element set of bit positions (0..9), must be evaluated in order for the genetic evolution to work. Thus the fitness function of the MGAC-CNS, has to evaluate a whole neighborhood of 2^3, or in general 2^n solutions.

The most proper thing to do this might be the exhaustive search method. However, this technique will consume 2^n fitness evaluations, for every evaluated solution (neighborhood) of the MGAC-CNS operator, or $5x7x2^n$ evaluations every time the operator is invoked.

In order to overcome this, we have used a neighborhood evaluation function that samples the specific neighborhood's solutions, by evaluating a small number m of bit combinations within the neighborhood under evaluation. In the simulations performed in this work we have used m=2. The bit combinations evaluated for each neighborhood are taken at random. After the evaluation of the samples, their fitness values are averaged to produce the final quality of the neighborhood under examination.

The above process is summarized in the following (see also Figure 2):

1. For the Nth time the MGAC-CNS is invoked:
2. Produce P-1 random parent vectors (neighborhoods) S_i, i=1..P-1, $S_i=(S_{i1}, S_{i2}, \dots, S_{in})$ (P is the no of genotypes in the MGAC-CNS population, e.g. P=5, n is the bits per neighborhood, e.g. n=3).
3. Inject the best neighborhood S_{best} of the previous (N-1) MGAC-CNS run as the Pth parent genotype.
4. Evaluate each S_i neighborhood:

4-1. Randomly produce m (e.g. m=2) samples of bit combinations for this neighborhood (e.g. 101, 011)

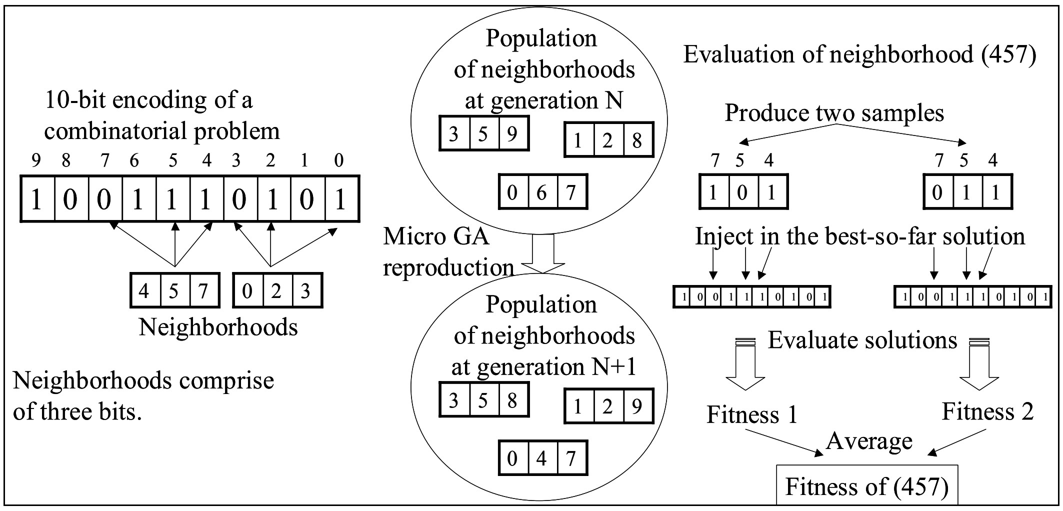
4-2. Inject the bits of each sample to the corresponding bits of the best-so-far main GA solution.

4-3. Calculate Fitness Fj (j=1..m) of the resulting genotype, using the main GA's fitness function.

4-4. Average Fj (j=1..m), to calculate the fitness of neighborhood S_i.

5. Mate and reproduce neighborhoods S_i, i=1..P, of the parent population, using common crossover and mutation operators, and produce the generation of offspring neighborhoods.
6. Continue for G generations (e.g. G=7)
7. If the MGAC-CNS finds a solution better than the best-so-far solution of the main GA, then the MGAC-CNS solution replaces the corresponding main GA solution.

One drawback of MGAC-CNS though is the fact that each neighborhood produced, is evaluated by a limited number of samples. This is adopted for practical computational reasons, because the exhaustive search of the neighborhood may consume quite a large number of fitness evaluations.



10-bit encoding of a combinatorial problem
9 8 7 6 5 4 3 2 1 0
1 0 0 1 1 1 0 1 0 1
4 5 7
0 2 3
Neighborhoods
Neighborhoods comprise of three bits.
Population of neighborhoods at generation N
3 5 9
1 2 8
0 6 7
Micro GA reproduction
Population of neighborhoods at generation N+1
3 5 8
1 2 9
0 4 7
Evaluation of neighborhood (457)
Produce two samples
7 5 4
1 0 1
7 5 4
0 1 1
Inject in the best-so-far solution
Evaluate solutions
Fitness 1
Fitness 2
Average
Fitness of (457)

Figure. 2. MGAC-CNS example on a combinatorial problem with a 10-bit encoding and neighborhoods of 3 bits. Micro GA searches the space of all possible neighborhoods. Each produced neighborhood is evaluated by producing two random samples (bit combinations), injecting them into the best-so-far genotype of the main GA, evaluate the two solutions and average the fitness values.

V. Simulation Results on the Unit Commitment Problem

In order to test the efficiency of the two MGAC operator variations, a difficult constrained combinatorial problem was selected, the Unit Commitment problem (UC).

The UC problem comes from the field of Power Systems and it is in fact a time scheduling problem. It involves determining the start-up and shut down schedules of thermal units to be used to meet forecasted demand over a future short term (24-168 hour) period. The objective is to minimize total production costs while observing a large set of operating constraints.

The total costs consist of a) Fuel costs, b) Start-up costs and c) Shut-down costs. Fuel costs are calculated using unit heat rate and fuel price information usually as :

$$FC = a + b \cdot P + c \cdot P^2 \qquad (2)$$

where P is the power output of a unit, and a, b, c are fuel cost coefficients. The power outputs of the committed units for every hour of the schedule are easily calculated by the λ-iteration algorithm [2], [6].

Start-up costs are expressed as a function of the number of hours the unit has been down. Here we have used the following formula:

$$SUC = \begin{cases} HSC, if\,(down \le cold\_start\_hours) \\ CSC, if\,(down > cold\_start\_hours) \end{cases} \qquad (3)$$

where HSC is the “Hot Start Cost” value and CSC is the “Cold Start Cost” value for the specific unit, “down” is the number of hours the unit has been down, and “cold_start_hours” is different for each unit. Shut-down costs are defined as a fixed dollar amount for each unit per shut-down, and is taken equal to 0 in this work. Thus the cost (objective) function to be minimized can be formulated as :

$$O(s) = \sum_{i=1}^{U}\sum_{j=1}^{H} FC_{ij}(s) + \sum_{i=1}^{U}\sum_{j=1}^{SU(i)} SUC_{ij}(s) \qquad (4)$$

where s is a specific solution, U is the number of units, H is the number of scheduling hours, FC_{ij} is the fuel cost of unit i at hour j, SU(i) is the number of start-ups of unit i during the schedule, and SUC_{ij} is the start-up cost of unit i at start-up j.

The constraints taken into account in this work, are: (a) System power balance (demand + losses + exports), (b) System reserve requirements, (c) Unit initial conditions, (d) Unit high and low MW limits (economic, operating), (e) Unit minimum-up time, (f) Unit minimum-down time.

For testing the MGAC operator we have used an instance of the UC problem presented in [6]. This instance includes 10 thermal power production units and a scheduling horizon of 24 hours. The unit data and the forecasted demand for the schedule are omitted for brevity and can be found in [6] .

For the application of GAs to the UC problem a simple binary alphabet was chosen to encode a solution. With the assumption that at every hour a certain unit can be either ON or OFF, an H-bit string is needed to describe the operation schedule of a single unit. In such a string, a '1' at a certain location indicates that the unit is ON at this particular hour, while a '0' indicates that the unit is OFF.

The main GA used a population of 50 genotypes, Roulette Wheel parent selection, multi-point crossover and bit mutation with adaptive application probabilities. Also some advanced operators were used just like in [6]. Additionally, the problem constraints were handled by adding penalty terms to the objective function, for every constraint violation, and applying the Varying Fitness Function technique [6], [12] for the gradual application of penalties through the GA run, just like in [6].

TABLE I. SIMULATION RESULTS OF 3 METHODS, ON THE UC PROBLEM, REPORTED IN [6].

units	Dynamic Programming	Lagrangian Relaxation	Simple GA					
	best solution	best solution	gener. limit	Mean No of Evaluat	success %	Best solution	worst solution	Differ. %
10	565825	565825	500	35,000	60%	565825	570032	0.74
20	-	1130660	1000	70,000	75%	1126243	1132059	0.51
40	-	2258503	2000	140,000	90%	2251911	2259706	0.34
60	-	3394066	3000	210,000	100%	3376625	3384252	0.22
80	-	4526022	4000	280,000	100%	4504933	4510129	0.11
100	-	5657277	5000	350,000	100%	5627437	5637914	0.19

TABLE II. SIMULATION RESULTS OF THE TWO MGAC VARIANTS ON THE UC PROBLEM.

units	GA with the MGAC-ARM operator						GA with the MGAC-CNS operator					
	gener limit	success %	Best solution	worst solution	Differ. %	Average solution	gener. limit	success %	best solution	worst solution	Differ. %	average solution
10	350	60%	565825	566764	0,166	566066	320	40%	565825	566977	0.203	566129
20	700	100%	1126242	1130521	0,379	1128534	640	80%	1126028	1131951	0.526	1129558
40	1400	100%	2250921	2255540	0.205	2253883	1280	100%	2251090	2256468	0.238	2254899
60	-	-	-	-	-	-	1920	100%	3376429	3386609	0.301	3381095
80	-	-	-	-	-	-	2560	100%	4501905	4510403	0.188	4506909
100	-	-	-	-	-	-	3200	100%	5623547	5634774	0.199	5630006

(The "Differ.%" figures are the percent difference between the best and worst solutions. A GA run is considered successful when it outperforms the corresponding Lagrangian Relaxation solution. All costs are in US$.)

The simulations included test runs for 10, 20, 40, 60, 80 and 100 unit problems. For the 20-unit problem the units were duplicated and the demand values were doubled. The data were scaled appropriately for the larger problems. The two GA-MGAC variations are compared to three other algorithms implemented and tested in [6]: a Dynamic Programming algorithm, a Lagrangian Relaxation algorithm, and a "simple" GA without the MGAC operator. In order to have a fair comparison, the runs are conducted on the basis of 'equal number of evaluations' for all GA algorithms. On the 10 unit problem, simple GA runs consume an average of 35,000 evaluations 25,000 (population 50 X 500 generations) of which are consumed by the simple GA and the rest 10,000 by custom hill-climbers. In order for the MGAC-ARM scheme to consume the same total number of evaluations, the generation limit was reduced to 350 generations (Table II), and for the MGAC-CNS to 320 generations.

For every problem and method, 20 independent runs have been performed as in [6]. The results are shown in Table II. Dynamic Programming was run only for the 10-unit problem, as for the larger problems the memory and cpu-time resources needed were prohibitive. The GA-MGAC-ARM variation was applied to problems of up to 40 units, as the number of possible neighborhoods that have to be enumerated and ranked rises to extremely high values for larger problems (13,160,160 neighborhoods for the 40-unit problem and 109,230,240 for the 60-unit one. The numbers represent not all possible neighborhoods, but all valid ones. Valid neighborhoods consist of 5units X 5hours, where the hour bits are contiguous and the same for all units).

From the results presented in Table II the following conclusions can be drawn:

For the 10-unit problem the two GA-MGAC schemes find the same solution as the other 3 algorithms with a success rate of 60% for the ARM variant, and 40% for the CNS variant, compared to 60% of the simple GA. Moreover, the worst solution produced by the two GA-MGAC schemes is much closer to the optimum than the corresponding worst solution of the simple GA, despite the reduced generation limits. This explains the low percent difference of 0.166% (ARM) and 0,203 (CNS). Thus, for the 10 unit problem ARM seems to slightly outperform the simple GA, in terms of average solution quality, while CNS seems to be slightly worse in terms of the success rate of finding the exact optimum, and slightly better in average solution quality than the simple GA. This could be due to the reduced generation limit of CNS.

For the 20-unit problem the GA-MGAC-ARM finds a solution slightly better than that of the simple GA, yet exhibiting better success rate and average solution quality, while the CNS variation discovers a new up-to-now best solution with a cost of 1126028, and exhibits similar performance compared to the simple GA. All of the 20 ARM runs produce solutions better than the best solution produced by the Lagrangian Relaxation method. Additionally, the worst solution is much closer to the best one, than the corresponding worst solution of the simple GA, and this leads to a smaller percent difference value of 0.379 (simple GA: 0.51). Here the ARM variant seems to clearly outperform the simple GA, while CNS is slightly worse and more or less even with simple GA, except for the new best solution it discovers.

For the 40-unit problem both GA-MGAC variants discover new up-to-now solutions with a cost of 2250921 (ARM) and 2251090 (CNS). Again, all 20 runs of both variants produce solutions better than that of the Lagrangian Relaxation method. Moreover, the worst solution of both variants is again much closer to the best one, than that of the simple GA, resulting again in a smaller percent difference value of 0.205 (ARM) and 0.238 (CNS). In this problem the two variants exhibit similar performances, but clearly outperform the simple GA in terms of the best and average solution quality.

For the 60, 80, and 100-unit problems the CNS variant seems to outperform the simple GA considering the best solutions found, exhibiting thus better in-depth search efficiency, but is slightly worse in terms of average solution quality (% difference). This could be due to the reduced generation limit of the GA-MGAC-CNS algorithm, that is mandatory in order to achieve the same total number of fitness evaluations.

From the above it is evident that both MGAC operator variations really enhance the GA performance on a difficult constrained combinatorial problem, like the UC problem presented. They are capable of discovering even better solutions than the best known ones for the specific instances of the UC problem used in this work. They also seem to enhance the robustness of GAs in consistently finding solutions close to the optimum. Also it is evident that the ARM variant is most suitable for relatively small problem scales (e.g. up to 40X24=960 bits problems), where it has a clear advantage over the simple GA, while the CNS variation is most suitable for large scale problems, where the ARM variation is not applicable. In large scale problems the CNS variation also seems to have a clear advantage over the simple GA, especially in discovering solution close to the global optimum.

VI. Conclusions

In this paper, a new hill climbing operator for genetic optimization of combinatorial problems has been presented, the Micro GA Combinatorial hill climbing operator (MGAC). Two variants of MGAC have been presented: the ARM and the CNS variants.

Both variants try to search combinatorial neighborhoods or subsets of the symbol strings that encode complete solutions of the main GA, keeping a part of the best-so-far solution constant, and allowing a subset (neighborhood) of it to change.

The performance of the two operator variations has been demonstrated by their application on a power systems scheduling problem, the Unit Commitment problem. The simulation results have shown that the MGAC operator is able to boost the performance of the main GA in difficult combinatorial problems, by improving its search efficiency and robustness.

More effort is also needed in applying the two MGAC operator variations to more combinatorial problems, in order to test if their performance and robustness can be generalized.

References

[1] D. H. Ackley, "Stochastic Iterated Genetic Hill Climbing," Ph.D. Thesis, Department of Computer Sciences, Carnegie Mellon University, Pittsburgh, PA, 1987.

[2] A. Bakirtzis, V. Petridis, S. Kazarlis, "A Genetic Algorithm Solution to the Economic Dispatch Problem," *IEE Proceedings - Generation, Transmission, Distribution*, Vol. 141, No. 4, July 1994, pp. 377-382.

[3] G. Dozier, J. Brown and D. Bahler, "Solving Small and Large Scale Constraint Satisfaction Problems using a Heuristic-based Microgenetic Algorithm," in *Proc. of the 1st IEEE Int. Conf. on Evolutionary Computation*, vol. 1, Piscataway, NJ: IEEE Press, pp. 306-311, 1994.

[4] S.Kazarlis, S.Papadakis, J.Theocharis and V.Petridis, "Micro-Genetic Algorithms as Generalized Hill Climbing Operators for GA Optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 3, June 2001, pp. 204-217.

[5] S. A. Kazarlis, "Micro-Genetic Algorithms As Generalized Hill-Climbing Operators for GA Optimization of Combinatorial Problems – Application to Power Systems Scheduling", *Proceedings of the the 4th Conference on Technology and Automation, October 2002*, Thessaloniki, Greece, pp. 300-305.

[6] S. A. Kazarlis, A. G. Bakirtzis, and V. Petridis, "A Genetic Algorithm Solution to the Unit Commitment Problem," *IEEE Trans. on Power Systems*, vol. 11, no. 1, pp. 83-92, Feb 1996.

[7] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization," in *SPIE Proceedings: Intelligent Control and Adaptive Systems*, pp. 289-296, 1989.

[8] J. A. Miller, W. D. Potter, R. V. Gandham, and C. N. Lapena, "An Evaluation of Local Improvement Operators for Genetic Algorithms," *IEEE Trans. on Syst., Man, Cybern.*, vol. 23, No. 5, pp. 1340-1351, 1993.

[9] M. Mitchell, J. Holland, and S. Forrest, "When will a genetic algorithm outperform a hill-climbing?," in J. D. Cowen, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, San Mateo, CA, 1994, Morgan Kaufmann.

[10] H. Muhlenbein, "How Genetic Algorithms really work: I. Mutation and hill-climbing," in R. Manner and B. Maderick, editors, *Parallel Problem Solving from Nature 2*, pp. 15-25, Elsevier, 1992.

[11] V. Petridis and S. Kazarlis, "Varying Quality Function in Genetic Algorithms and the Cutting Problem," in *Proceedings of the 1st IEEE Conference on Evolutionary Computation* (vol. 1). Piscataway, NJ: IEEE Press, 1994, pp. 166-169.

[12] V. Petridis, S. Kazarlis, and A. Bakirtzis, "Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 28, Part B, No. 5, October 1998, pp. 629-640..

[13] N. J. Radcliffe and P. D. Surry, "Formal Memetic Algorithms," in *Proceedings of the 1st AISB Workshop on Evolutionary Computing* (AISB '94), T. C. Fogarty, Editor, Springer-Verlag, pp. 1-16, 1994.

[14] J-M. Renders and H. Bersini, "Hybridizing Genetic Algorithms with Hill-Climbing Methods for Global

Optimization: Two Possible Ways," in *Proc. of the 1st IEEE Int. Conf. on Evol. Computation*, vol. 1, Piscataway, NJ: IEEE Press, 1994, pp. 312-317.