When Local Search Goes With the Winners

Bertrand Neveu and Gilles Trombettoni
{neveu, trombe}@sophia.inria.fr
COPRIN Project, CERMICS-I3S-INRIA, 2004 route des lucioles,
06902 Sophia Antipolis cedex, B.P. 93, France

Abstract

A new combinatorial optimization meta-heuristic, called *"Go With the Winners"* (`GWW`) has been proposed by Dimitriou and Impagliazzo in 1996. `GWW` manages several configurations at the same time, uses a threshold to eliminate the worst configurations, and includes a randomization step which allows a uniform distribution of the visited configurations in every sub-problem.

This paper proposes to replace the randomization step of `GWW` by a local search in order to favor better solutions during the search.

An instance of this hybrid algorithm, called `GWW-grw`, is proposed. A special attention is paid to make it efficient on realistic problems. We compare different ways to lower the threshold in an adaptive way. We add only one parameter for embedding local search in the existing scheme. We study how, and in which order, to tune the parameters.

An experimental approach has been performed on hard instances, encoded as weighted MAX-CSPs, of graph coloring problems (DIMACS challenge) and frequency assignment problems (celar). The best known bounds have been found for all the instances.

Keywords: combinatorial optimization, incomplete algorithms, `GWW`, celar problems, graph coloring

1 Introduction

Our work is based on the *"Go With the Winners"* algorithm (`GWW`) presented in [7]. `GWW` explores different configurations of the search space at the same time by managing a population of solutions called *particles*. At the beginning, particles are randomly distributed in the search space and a threshold is placed at the cost of the worst particle. The following phases are iteratively performed until no particle remains under the threshold (We consider a minimization problem here.):

1. **Redistribution:** The particles over the threshold are "redistributed" on the others (hence the name of the algorithm): a redistributed particle is replaced with a duplicate of another particle randomly chosen under the threshold.

2. **Randomization:** This phase is performed for every particle and intends to "separate" particles which are at the same location due to the redistribution phase. Several randomization phases can be found in the litterature. They are all based on a random walk of specified length. Every step of the random walk goes from a configuration to a neighbor while trying to remain under the threshold.

3. The value of the threshold is lowered by 1.

When the problem has good properties (see Section 2), GWW is theoretically able to find a best solution in polynomial time with a high probability. However, in practice, first experiments made on celar and graph coloring instances give worse performance results than a standard Metropolis algorithm [13] (a simulated annealing with a constant temperature).

In the randomization phase of the GWW algorithm, better configurations are not preferred and a particle can indifferently go up or down. Thus, the need to make progress in GWW is due only to the management of the threshold. In this article, we experimentally show that GWW can be made more efficient when replacing the randomization phase by a local search. This gives the GWW-LS algorithm which will be described in details further. The main contributions are the following:

- In previous works, GWW has been applied on theoretical models: on the clique problem and on the graph bisection problem[1]. All the corresponding versions of GWW assumed that the range of possible costs included a small number of integers, so that the threshold delta was 1 at each iteration. This is not the case for numerous problems such as the weighted MAX-CSP benchmarks presented in this paper. For example, the `celar7` benchmark typically includes configurations whose cost ranges from 343000 to 2.10^7. Therefore, this article proposes several ways to lower the threshold in realistic problems, and perform experiments to compare them each other.

- GWW-LS manages several parameters: the number B of particles, a parameter T used to lower the threshold, the length S of the walk, and additional parameters involved in the selected local search (e.g., the length of a tabu list, or a "temperature" allowing the search to escape from local minima). We propose an instance of the GWW-LS scheme, called `GWW-grw` (GWW with *greedy-random walk*), which limits the number of additional parameters to 1. In order to minimize the effort required for tuning the parameters, experiments show a heuristic policy to tune the parameters of `GWW-grw`.

- Experiments are performed on very hard instances of frequency allocation and graph coloring problems. `GWW-grw` can find the best (known) bounds for all of them and comparisons with GWW and Metropolis are given.

[1]Split a graph into two vertex-induced sub-graphs with the same size such that a minimum number of edges link nodes from the two sub-graphs.

Section 2 gives a brief history of GWW. Section 3 details the GWW-LS. The GWW-grw instance of GWW-LS is described in Section 4. Experiments are reported in Section 5. Section 5.2 presents several ways to manage the threshold in realistic problems. Section 5.3 gives a first heuristic policy to tune the four parameters of GWW-grw in practice. Sections 5.4 and 5.5 show the performance of GWW-grw and report comparisons with GWW and Metropolis.

2 From GWW to GWW-LS

Before presenting the GWW algorithm, we define a *search graph* as an abstraction of a combinatorial optimization problem. The nodes of a search graph are the different configurations (i.e., potential solutions) and edges link two configurations which are "neighbors", i.e., for which a small change allows to pass from a configuration to the other. We assume that the value, or cost, of every node is an integer. The goal of combinatorial optimization is to find a node of minimum cost.

The "Go with the winners" algorithm has been first introduced in [1]. This algorithm has been designed to find a deep leaf in a tree. At the beginning, several particles are placed at the root of the tree. In the randomization phase, every particle is moved to a child node which is chosen randomly. In the redistribution phase, the particles which have reached a leaf are redistributed: they are replaced with a duplicate of another non leaf particle randomly chosen. The process stops when all the particles are placed at a leaf. The authors have determined the number of particles which is necessary to find a deepest node with a high probability depending on a measure of the tree imbalance.

The GWW algorithm discussed in this paper and informally presented in the introduction, is presented in [7]. It is a modification of the first version so that it can be used directly for combinatorial optimization. The tree of the first version occurs in any optimization problem when managing a threshold. Indeed, the search graph can be hierarchically divided into several sub-graphs according to their costs. More precisely, a node of the tree is made of a search sub-graph whose vertices have a cost at or under a threshold. The top of this tree contains the whole search graph (when the threshold is the highest). Lowering the threshold divides a search graph into several (disconnected) connected components which form a hierarchy between a tree node and its sons. The fact that two regions become disconnected when the threshold decreases means that no random walk (staying at or under the threshold) can move a particle from a region to the other.

Dimitriou and Impagliazzo relate in [8] the performance of GWW to combinatorial properties of the problem. Especially, they prove that two sufficient conditions make a problem solvable in polynomial time with a high probability:

- The number of children of a tree node must be polynomially bounded. This means that decreasing the threshold (by 1) does not make appear an exponential number of disconnected regions. This guarantees convergence in polynomial time if every disconnected region has a particle.
- *Local expansion* holds, that is, every search sub-graph has a "sufficient" number of neighbors. Thus, a sufficiently long random walk in a given region ensures a uniform distribution of the particles which are inside and does not restrict the particles into a small portion of the region. The random redistribution and the

randomization phase ensure that every connected component receives a number of particles proportional to its size.

The first condition highlights the importance of the number B of particles. The second one is related to the length S of a random walk.

Different randomization phases for `GWW` are described in the litterature[2]:

- In [7], a random walk of length 1 is performed at each step: for every particle with cost i, all the neighbors are visited and one is chosen among those having a cost $i-1$ (or $i+1$ in a maximization problem).
- In [8], a random walk of length S is proposed. This walk alternates nodes of cost i and cost $i-1$.
- [3] and [6] present a simplified version where the random walk of lenght S has no other restriction than remaining at or under the threshold. That is, a neighbor is accepted if its cost is strictly less than $i-1$.

This article introduces the `GWW-LS` algorithm for which the randomization phase of `GWW` is replaced by a local search where every step of the walk remains at or under the threshold. The difference between the two approaches is that a local search tends to make progress whereas a random walk does not. This tends to faster bring down the population, but violates the uniform distribution of particles in a connected region (distribution on which the local expansion property is built). However, the good experimental results which are obtained using `GWW-LS` make us believe that other sufficient conditions for the convergence in polynomial time with a high probability exist. A possible hypothesis is discussed in the conclusion.

3 Description of `GWW-LS`

`GWW-LS` is very close to the existing `GWW` algorithm. However, `GWW-LS` differs from `GWW` in two points:

- *The descent of the threshold is adaptive.*
  Section 5.2 discusses and compares several ways to implement the `Lower` function used to lower the threshold. Unfortunately, this realistic management implies the addition of a new parameter T to the `GWW-LS` scheme.
- *The walk is not random.*
  The two `for` loops of Algorithm 1 describe a walk: every particle makes a walk of length S. As already mentioned, a walk of length S is not specific to `GWW-LS` and is also used in previous versions of `GWW` (see [3, 6]). However, the major difference lies in the `step` function.

In `GWW`, this function returns a neighbor of the p particle configuration. More precisely, all[3] the neighbors are visited in a random order until one is selected which is under or at the threshold. If all the attempts fail, the current configuration is returned, that is,

[2]In the initial version [1], a particle is simply moved to a child.

[3]or a bounded number of

the particle does not move at this step. In GWW-LS, the function step aims at making progress while escaping from local minima.

```
algorithm GWW-LS (B: number of particules; S: walk length, T: threshold parameter; WP: walk parameters): configuration
    Generate $B$ random configurations and place every particle on one configuration
    Store the particles in an array Particles
    threshold ← $\infty$
    Loop
        threshold ← Lower(threshold, $T$, Particles)
        if Best(Particles) > threshold then return (Best-found) /* the best configuration found during search */
        Redistribute(Particles) /* Move every particle having a cost more than threshold to the configuration of an alive particle randomly chosen */
        for every particle $p$ in Particles do
            for $i$ from 1 to $S$ do
                neighbor ← step($p$, $WP$)
                Move(p,neighbor) /* A move of $p$ to a new configuration */
            end.
        end
    end.
end.
```

Algorithm 1: The GWW-LS algorithm

4 From GWW-LS to GWW-grw

We need to design at least one algorithm deduced from the GWW-LS and show its efficiency in practice. We tried several standard classical local search algorithms to improve GWW. We obtained good results with a Metropolis algorithm (accepting surely a neighbor with the same or a better cost, and accepting a worse neighbor with a probability depending on a given "temperature"). However, an important work has led to simplify it in two ways: minimizing the number of parameters, and finding how to tune them more quickly, while increasing efficiency.

The classical GWW manages two parameters: the number B of particles and the length S of the random walk. In practice, a third parameter T needs to be added allowing the threshold to not decrease too smoothly. Our new algorithm GWW-grw (GWW with *greedy-random walk*) adds only one parameter N. This unique parameter is used by a particle to progress downto the best solutions while being able to escape from local minima. The parameter N is the maximum number of neighbors which are visited by a particle during the step function, that is, when a particle moves from a configuration to another. Based on a current configuration x, the computation of the new configuration returned by the step function of GWW-grw is performed as follows:

1. one neighbor x' among the N ones has a cost better or equal than the cost of x $\Rightarrow$ **Return** x'

2. no neighbor among the N ones has a cost better or equal than the cost of x **and** one neighbor x'' among the N ones is at or under the threshold $\Rightarrow$ **Return** x''

3. `the` N `visited neighbors are above the threshold` $\Rightarrow$ **Return** x

The first case radically differs from the standard `GWW`. It reproduces the behavior of the hill-climbing and GSAT [19] algorithms: a neighbor is accepted only when its cost is better or equal than the current one. The last two cases indicate how to select a configuration when no visited neighbor has been accepted. They follow the random selection used by most of the algorithms based on a threshold, e.g., `GWW`, the *threshold accepting* and the σ-*algorithm* discussed in Section 6. In particular, the second case allows particles to exit local minima. Some particles may go up very high during the search, but too bad configurations are discarded by the threshold.

Thus, parameter N avoids visiting all the neighbors and yields a trade-off between making progress and exiting local minima. To sum up, `GWW-grw` manages the following parameters:

- A number B of particles used to explore the search space in parallel: particles should be uniformly distributed within the connected components appearing in the search graph when the threshold is lowered.
- A length S of the walks used to separate the particles after grouping, and to allow particles to progress downto the best solutions.
- A parameter T used to manage the threshold in real problems (see Section 5.2).
- An additional parameter N which has a crucial role. First, this is necessary to limit the number of visited neighbors in problems for which it is very big and discards an exhaustive search. Second, N must be sufficiently large to allow particles to go down onto better solutions (in a greedy way). Third, N must be sufficiently small to allow the particles to exit local minima in a random way.

Complexity of `GWW-grw`

The worst-case complexity of `GWW-LS` or `GWW-grw` remains the same as the one of `GWW`, that is, $0(B \times S \times N \times nT)$, where nT is the number of times the threshold is lowered, B is the number of particles and S is the walk length.

5 Experiments

This section shows experimental results. We select a way to manage the threshold. We give guidelines to tune the four parameters in practice. Finally, `GWW-grw` is compared with existing algorithms: the standard `GWW` and Metropolis.

5.1 Benchmarks, encoding, implementation

We have performed experiments on difficult instances issued from two categories of problems and all encoded as weighted MAX-CSPs problems: graph coloring instances proposed in the DIMACS challenge ten years ago [17], and frequency assignment problems[4].

[4]Thanks to the "Centre d'électronique de l'Armement".

Graph coloring instances

We have selected three difficult graph coloring instances from the two most difficult categories in the catalogue: the `le450_15c` with 450 nodes and 16680 edges, the `le450_25c` with 450 nodes and 17425 edges, and the more dense `flat300_28` instance with 300 vertices and 21695 edges. In all instances, a best solution is hidden with respectively, 15, 25 and 28 colors. In this paper, graph coloring instances are encoded as MAX-CSP instances: variables are the vertices in the graph to be colored; the number d of colors with which the graph must be colored yields domains ranging from 1 to d; vertices linked by an edge must be colored with different colors: the corresponding variables must take different values. Coloring a graph in d colors amounts in minimizing the number of violated constraints and finding a solution with cost 0.

Celar frequency assignment instances

We have also selected the three most difficult instances of the frequency assignment problems: the `celar6`, the `celar7` and the `celar8` problems. These instances are realistic since they have all been built from different sub-parts of a real problem. The `celar6` has 200 variables and 1322 constraints; the `celar7` has 400 variables and 2865 constraints; the `celar8` has 916 variables and 5744 constraints.

The variables are the frequencies to be assigned a value which belong to a predefined set of allowed frequencies (domain size about 40). The constraints are of the form $|x_i - x_j| = \delta$ or $|x_i - x_j| > \delta$ (to avoid interferences). Our encoding is standard and creates only the even variables in the CSP along with only the inequalities[5].

The objective function to be minimized is a weighted sum of violated constraints. The constraints belong to four categories with different costs of violation:

- Constraints of `celar8` have a weight 1, 2, 3 or 4.
- Constraints of `celar6` have a weight 1, 10, 100 or 1000.
- Constraints of `celar7` have a weight 1, 100, 10000 or 10^6.

Neighborhood

The usual definition of neighborhood used for CSPs is chosen here: a new configuration x' is a neighbor of the current configuration x if both have the same values, except for one variable which takes different values in both configurations. Moreover, following Minton's heuristic [16], the current value of that variable participates to a conflict in the configuration x, that is, it violates at least one constraint.

Implementation

All the algorithms have been developed in a same software [18]. Our platform is implemented in C++ and all the tests have been performed on a `PentiumIII 935 Mhz` with a Linux operating system. All the algorithms belong to a hierarchy of classes which aims at sharing code, so that fair comparisons can be made between them.

[5]A bijection exists between odd and even variables. A simple propagation of the equalities allows us to deduce the values of the odd variables.

5.2 Threshold management

Our first experiments showed that lowering the threshold by a cost 1 at each iteration was irrelevant on celar problems for which the cost range is very large. Even on graph coloring whose objective function is more "flat", we obtained even better results by paying attention to the threshold management. Several possible threshold managements have then been designed. They differ from each other in the delta (in terms of cost) to be subtracted to the current threshold at every iteration:

1. $\Delta_{blind} = 1 + \text{threshold} \times T_{blind}$
2. $\Delta_{s-blind} = 1 + \text{dist}(B) + \text{threshold} \times T_{s-blind}$
3. $\Delta_{adaptive} = 1 + \text{dist}(i) \quad (= 1 + \text{dist}(B) + 100\%(\text{cost}(B) - \text{cost}(i)))$
4. $\Delta_{best} = 1 + \text{dist}(B) + T_{best}(\text{cost}(B) - \text{cost}(1))$
5. $\Delta_{user} = 1 + \text{dist}(B) + (\text{threshold} - \texttt{user\_bound}) \times T_{user}$

T_{blind}, $T_{s-blind}$, T_{best}, T_{user} or i is the parameter to be ruled to make the threshold more or less smooth. The first four parameters are ratios; dist(i) is the distance (i.e., cost difference) between the threshold and the i^{th} best particle; thus, B is the worst (highest) particle and dist(i) is the distance (cost difference) between the threshold and particle B. cost(j) is the cost of the j^{th} best particle.

The first delta tested was Δ_{blind}. The other deltas are adaptive and take into account the way the particles go down. They question two characteristics of Δ_{blind}: Δ_{blind} may be too small at the beginning and too large at the end. Indeed:

- At the first threshold modifications, the particles go down a lot since moving to better configurations is very easy and even the worst particle may be very far under the threshold.

  The dist(B) component of the four last deltas is mainly usefull in this case.

- At the end, when the threshold approaches the best solution, the delta remains large if the best solution cost is even greater than 0 (this is the case for the celar problems).

  `user_bound` is a second parameter necessary for tuning Δ_{user}. It corresponds to a lower bound specified by the user depending on the problem (e.g., 3389 for `celar6` since this bound has been proven by complete techniques). Δ_{user} is tested to check the interest of having a smooth threshold at the end.

The conclusions of the tests reported below are clear. First, it is generally interesting to force the threshold to be under the worst particle. The second conclusion contradicts our first intuitions: paying attention to the way the threshold decreases at the end is not fruitfull. That is why we have adopted the simple delta management $\Delta_{s-blind}$. Details are given below.

$\Delta_{adaptive}$ turned out to quickly converge to 1 while being far above the best solutions. Indeed, when making progress becomes difficult, the particles are not uniformly distributed (in terms of cost) between the threshold and the best particle, more and more particles remain at the threshold and dist(i) becomes null.

To overcome this drawback, Δ_{best} is based on the largest possible adaptive distance since $\text{cost}(B) - \text{cost}(1)$ becomes null only when `GWW-LS` terminates.

Finally, the bad results obtained by a management based on Δ_{best} against the simple Δ_{blind} on instances with non flat criteria have led us to correct the latter, and we have designed and adopted $\Delta_{s-blind}$. The series of tests leading to this choice are gathered in Table 1.

	time	**blind**	**adaptive**	**best**	**s-blind**	**user**
`le450_15c`	103	61 (44)	4.2 (42)	0.9 (42)	**0.2 (41)**	–
`le450_25c`	70	11.4 (29)	10.2 (32)	10.1 (31)	10.1 (31)	–
`flat300_28`	112	3.7 (48)	3.4 (48)	3.6 (48)	3.7 (49)	–
`celar8`	140	303 (103)	335 (123)	306 (99)	**289 (106)**	303 (104)
`celar6`	102	3504 (49)	3515 (64)	3557 (43)	**3451 (49)**	3537 (41)
`celar7`	135	372125 (138)	627774 (152)	422616 (135)	377214 (141)	**370784 (138)**

Table 1: Threshold management. The cells report the average solution cost (weighted sum of violated constraints) obtained by `GWW-grw` in a given running time in seconds (second column); into parentheses, the average number of threshold modifications. The best results are bold-faced.

For every instance, 10 trials have been performed and average values are reported in the table[6]. The celar instances have been handled with $B = 50$ ($B = 200$ for `celar6`), $S = 200$, $N = 40$. The graph coloring instances have been solved with $B = 20$, $S = 2000$ and several values for N.

First, the tests on `le450_25c` and `flat300_28` are not informative. Second, $\Delta_{adaptive}$ is often the worst and Δ_{user} generally gives worse results than $\Delta_{s-blind}$. Third, $\Delta_{s-blind}$ is never bad and is often the best.

5.3 Tuning the parameters

An advantage of `GWW-grw` is that each of its four parameters seems to have a specific role. However, three among the four parameters are a priori not easy to be tuned. Indeed, reducing T or increasing the value of B and S always leads to better bounds with a worse performance. It is indeed not straightforward to select a value implying an asymptotic behavior or to decide which parameter to favor to improve bounds. However, a very fine tuning of the parameters seems not to be useful in practice, and we have already obtained promising tuning procedures. Our experiments and the literature lead to tune the parameters in the following heuristic way (in order):

1. **T:** The asymptotic behavior of T is the easiest to observe. In our experiments, the ratio T_{sblind} and T_{blind} were always chosen between 0.5% and 2%; the ratio T_{best} between 0.5% and 10%. Bigger ratios lead to very bad bounds and smaller ratios do not improve the bound while wasting a lot of time. Therefore, starting with a ratio with value 1%, a few trials (typically 2-4) are sufficient to select an acceptable value[7].

[6] A trial-and-error process is necessary to select the adequate value of a threshold parameter leading to a given average time...

[7] Small default values for B, S and N can be chosen at this step, e.g., $B = 20$, $S = 200$, $N = 50$.

2. **N:** Experiments reported below show that the value of N has a significant impact on the performance. Interesting in practice, two or three tests with different small values of B and S are sufficient to determine an acceptable value of N for a given instance.

3. **S:** Following the study performed in [6], tests described further would allow us to determine a walk length sufficiently long to uniformly explore the search graph sub-components.

4. **B:** Once the previous parameters have been tuned, B can be increased until a good solution is obtained.

Of course, this study should be considered a first approach. Our experiments make us believe that tuning T and N is easy. A deeper understanding of the `GWW-grw` behavior should explain why our first attempt for tuning S and B is acceptable and should lead to a more accurate tuning policy.

Tuning N

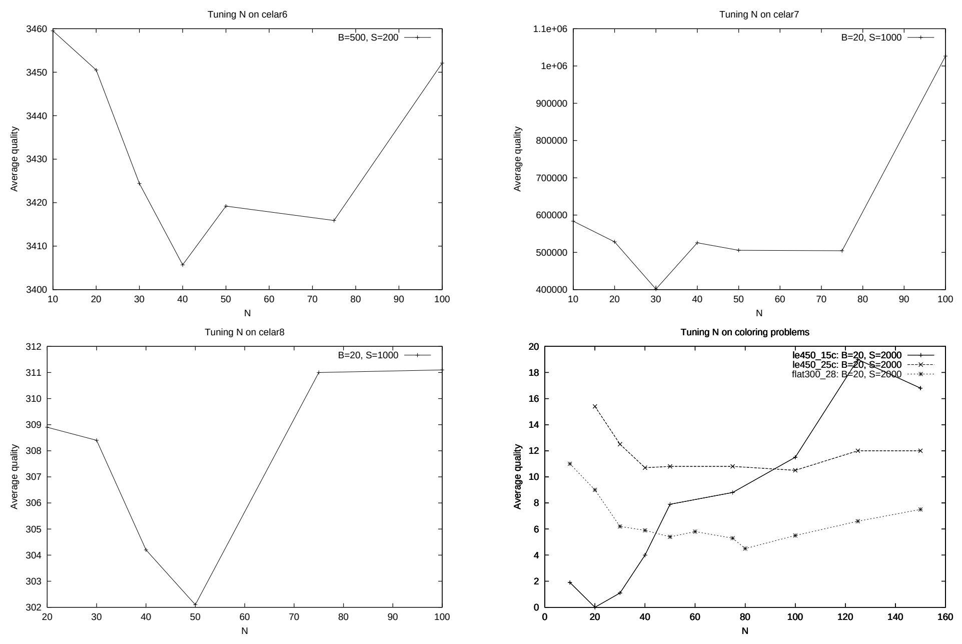

Figure 1: Tuning the N parameter. The values chosen for parameters B and S are indicated in the figures.

Figure 1 shows the curves obtained for tuning the parameter N on the chosen instances. The value of N is given in abscissa and the ordinates give the cost of the best solution (average on 10 tries). The conclusions are the following:

- It is not difficult to tune N since the curves own a minimum. Indeed, if N is too small, particles cannot make progress; if it is too large, particles cannot exit from local minima.

- Tuning N has a significant impact on performance. In particular, it allows GWW-grw to always color in 15 colors le_450_15c in 2 minutes. Only the celar7 instance does not seem to be very sensitive to this parameter.

We should mention that, for all tested instances, the curves obtained with different parameters B and S have roughly the same shape and give the same "best" value for N. This seems to mean that the value of N is intrinseque to a given instance.

Tuning S

We have applied the technique described in [6] for the standard GWW. In that article, tests have been performed with different values of S but a constant value for $B \times S$: the value of S giving the best solution (on average) is selected. This approach is justified by the following intuition. In GWW, S must be sufficiently long to allow the particles to be uniformly distributed into the search graph sub-components. Thus, when this "minimum" length has been reached, there is no interest to diffuse more.

The situation is more complicated for GWW-grw where S is used to diffuse the particles within the connected components but also to make progress downto the best solutions. Also, the technique described above has been applied for GWW which has only two parameters B and S, whereas our threshold management adds a third (monotonic) parameter T (assuming that N has been already tuned and fixed).

Anyway, we have applied this tuning approach to GWW-grw, and obtained two kinds of curves (not reported here due to a lack of space):

1. For the celar instances, curves give the best cost when S is comprised between 200 and 400, and the same value is obtained for different values of $B \times S$. This leads to adopt a large value B (several thousands) and a small value S (300) in the following tests.

2. For the graph coloring instances, the opposite behavior has been observed. The curves monotically decrease so that the best results are always obtained with a little number of particles, e.g. 20, and a long walk.

These tests could be a good indication of the relevance of GWW-grw. The interest for managing several particles in parallel is clear in the first case, whereas a local search could obtain better results in the second case.

5.4 GWW-grw performance

Table 2 reports the best bounds obtained by GWW-grw on the chosen benchmarks.

The flat300_28 instance has never been colored in 28 colors. To our knowledge, only three incomplete algorithms succeeded in coloring it in 31 colors: the distributed version of the Impasse algorithm [17], a tabu search [9], and a heuristic mixing a genetic algorithm and local search [11].

[5] and [15] report complete algorithms based on graph-based decomposition and dynamic programming. The other two instances are really challenging and have never been proven by a complete algorithm[8]. The algorithm in [14] is based on a very tuned

[8]This can be explained by the size of celar8 and by the criterion of celar7.

	Best algo	bound	time	GWW-grw	time	success	B, S, N
le450_15c	[17, 2]	15 col	min	15 col (0.2)	1.1 min	8/10	20,2000,20
le450_25c	[17]	25 col	min	25 col (1.4)	2.3 h	1/10	20,800000,100
flat300_28	[17, 9, 11]	31 col	hours	31 col (1.3)	5.9 min	2/10	20,20000,80
celar6	[5, 15, 21, 14]	3389	min/h	3389 (3405.7)	9.2 min	4/10	500,200,40
celar7	[21, 14]	343592	min	343598 (345941)	5.7 h	1/10	5000,300,50
celar8	[21, 14]	262	min	262 (267.4)	33.7 min	2/10	1000,200,30

Table 2: Results on benchmarks. The results of best known algorithms are reported in the left side of the table; the results of GWW-grw in the right side.

genetic algorithm with a mutation operator encoding a MIP. The algorithm described in [21] is a local search taking the conflicts into account to select neighbors.

Table 2 reports the impressive results obtained by GWW-grw. The best known bounds are obtained for all the tested instances. Furthermore, the time required by GWW-grw is often very satisfactory. To our knowledge, it is among the quickest algorithms for solving flat300_28, le450_15c and celar6. For the three other instances, the time required by GWW-grw is greater (hours against minutes for the best known algorithms). However, note that GWW-grw uses no technique or encoding which is specific to the type of problem, as opposite to algorithms described in [17, 14].

5.5 Comparing GWW-grw, GWW, and Metropolis

Table 3 gathers comparisons with Metropolis and GWW on the chosen instances. The column GWW' reports the results of GWW with the same threshold management as for GWW-grw. Our Metropolis is standard. It is implemented in the same software [18] and share most of the code with GWW-grw and GWW. It starts from a random configuration and a walk of length S is performed as follows:

- A new neighbor is accepted if its cost is better or equal than the current one.
- A new neighbor the cost of which is worse than the current one may be accepted with a probability depending on a given "temperature". This temperature is constant for Metropolis [4].
- When no neighbor is accepted, the configuration is not changed at the current iteration.

The other classical local search algorithms were not competitive with Metropolis on the chosen instances.

All the instances have been implemented with the same neighborhood. Of course, changing also the neighborhood implementation may change the gap between algorithms on a given instance, e.g., following Minton's heuristics also for the value choice improves the behavior of Metropolis on flat300_28.

The results of our tests can be interpreted as follows. First, the results of the pure GWW algorithm (with a threshold lowered by 1 at every iteration) are very bad and are not reported in the table. Second, GWW' gives bad results on graph coloring instances and is always worse than GWW-grw. Results are not so bad on celar problems, especially for celar8.

	Time/trial (min)	**Metropolis**	GWW'	GWW-grw
`le450_15c`	2	5.9 (2)	536 (410)	0 (0)
`le450_25c`	14	3.1 (2)	17.1 (14)	4 (3)
`flat300_28`	9	0.9 (0)	6.6 (6)	1.3 (1)
`celar6`	14	5048 (3906)	3648 (3427)	3405 (3389)
`celar7`	6	$6\,10^6$ $(2.9\,10^6)$	583278 (456968)	368825 (344317)
`celar8`	50	410 (300)	276 (265)	267 (262)

Table 3: Comparisons on benchmarks. Each cell contains the average cost of the solutions obtained (on 10 trials); the best cost appears into parentheses.

Metropolis has obtained bad results on celar problems. For `celar6` and `celar7`, it can be explained by the big violation cost associated to the constraints and for which only an adaptive temperature could be adequate. It seems better on graph coloring instances. It is not bad on `le450_15c` and is the best on `le450_25c` and `flat300_28`, where it slightly outperforms `GWW-grw`.

To conclude, `GWW-grw` is the best for 4 of the 6 instances and seems to always give good results (on the six presented instances). Although not reported in the tables above, the standard deviation of the costs obtained by `GWW-grw` is generally very small, which indicates that it is "robust". Finally, the results seem to confirm that using `GWW-grw` is relevant when the user can easily find a walk length S minimizing the cost while maintaining constant $B \times S$ (see end of Section 5.3).

6 Related Work

`GWW` can be viewed as a simplified version (no crossover operator) of a *genetic algorithm* [12]. The *threshold accepting* algorithm [10] and σ-algorithm [3] are local search heuristics based on a threshold. In both algorithms, a random walk is performed. At each move, the threshold can go down with a given probability. The *clustering* technique is mainly used for global optimization (over the reals) [20] where it obtains very good results. It manages a population and mechanisms are provided for avoiding the grouping of "particles".

7 Conclusion

We have designed a hybrid algorithm called `GWW-LS` introducing local search into the Go With the Winners algorithm. An instance of this scheme, called `GWW-grw`, has been defined. `GWW-grw` manages 4 parameters and a first attempt to tune them has been described. The management of the threshold for realistic problems has been studied. The maximum number N of visited neighbors allows `GWW-grw` to make progress downto the best solutions while exiting local minima. It is crucial in practice and seems to be able to be tuned independently.

`GWW-grw` has obtained very good results on graph coloring and celar problems. Only a few other optimization heuristics can reach these bounds on the tested instances. `GWW-grw` uses no technique or encoding which is specific to the type of problem.

Of course, `GWW-grw` should be tried on other types of problems, e.g., on instances containing hard constraints.

The good experimental results obtained by `GWW-grw` seem to confirm that the idea used by genetic algorithms consisting in exploring the search space in parallel is good. However, the fact that `GWW-grw` has no cross-over operator leads to several advantages:

- The algorithm is easier to be tuned.
- Since the neighborhood aspect of local search is preserved, incremental data structures can be used when passing from a configuration to its neighbor.
- The algorithm can be developed on a same plateform as local search algorithms.

As mentioned at the end of Section 2, the greedy descent of `GWW-grw` violates the uniform distribution of particles in a connected region (distribution on which the local expansion property is built). Therefore an interesting issue is to theoretically or experimentally establish another sufficient condition for the success of `GWW-grw`, e.g., the uniform distribution of the particles *at every cost level* of the connected components. This would indicate the interest of using a local search within `GWW-LS` for which particles are not trapped into local minima.

References

[1] D. Aldous and Vazinari U. Go with the winners algorithms. In *Proc. STOC*, 1994.

[2] Massimiliano Caramia and Paolo dell'Olmo. A fast and simple local search for graph coloring. In *Proc. of WAE'99*, number 1668 in LNCS, pages 316–329, 1999.

[3] Ted Carson and Russell Impagliazzo. Experimentally determining regions of related solutions for graph bisection problems. In *IJCAI, workshop ML1: machine learning for large scale optimization*, 1999.

[4] D. T. Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, (46):93–100, 1990.

[5] S. de Givry, G. Verfaillie, and T. Schiex. Bounding the optimum of constraint optimization problems. In *Proc. CP97*, number 1330 in LNCS, 1997.

[6] Tassos Dimitriou. Characterizing the space of cliques in random graphs using "go with the winners". In *Proc. AMAI*, 2002.

[7] Tassos Dimitriou and Russell Impagliazzo. Towards an analysis of local optimization algorithms. In *Proc. STOC*, 1996.

[8] Tassos Dimitriou and Russell Impagliazzo. Go with the winners for graph bisection. In *Proc. SODA*, pages 510–520, 1998.

[9] Raphaël Dorne and Jin-Kao Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In I.H. Osman S. Voss, S.Martello and C. Roucairol, editors, *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer Academic Publishers, 1998.

[10] G. Dueck and T. Scheuer. Threshold accepting : a general purpose algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[11] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[12] J. Holland. Adaptation in natural and artificial systems, 1975.

[13] S. Kirpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[14] A. Kolen. A genetic algorithm for frequency assignment. Technical report, Universiteit Maastricht, 1999.

[15] A. Koster, C. Van Hoesel, and A. Kolen. Solving frequency assignment problems via tree-decomposition. Technical Report 99-011, Universiteit Maastricht, 1999.

[16] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflict: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[17] Craig Morgenstern. Distributed coloration neighborhood search. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 335–357. American Mathematical Society, 1996.

[18] Bertrand Neveu and Gilles Trombettoni. Une bibliothèque de recherche locale pour l'optimisation combinatoire. In *Actes de ROADEF 2003*, 2003.

[19] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AI'92*, pages 440–446, 1992.

[20] A.A. Törn. Global optimization as a combination of global and local search. In *Proc. of Computer Simulation vs Analytical Solutions for Business and Economic Models*, pages 191–206, 1973.

[21] Christos Voudouris and Edward Tsang. Solving the radio link frequency assignment problem using guided local search. In *Nato Symposium on Frequency Assignment,Sharing and Conservation in Systems(AEROSPACE)*, 1998.