RAIRO-Oper. Res. 53 (2019) 1083–1095
https://doi.org/10.1051/ro/2018048

RAIRO Operations Research
www.rairo-ro.org

APPLICATION OF THE "DESCENT WITH MUTATIONS" METAHEURISTIC TO A CLIQUE PARTITIONING PROBLEM

Olivier Hudry[1,*]

Abstract. We study here the application of the "descent with mutations" metaheuristic to a problem arising from the field of classification and cluster analysis (dealing more precisely with the aggregation of symmetric relations) and which can be represented as a clique partitioning of a weighted graph. In this problem, we deai with a complete undirected graphe G; the edges of G have weights which can be positive, negative or equal to 0; the aim is to partition the vertices of G into disjoint cliques (whose number depends on G) in order to minimize the sum of the weights of the edges with their two extremities in a same clique; this problem is NP-hard. The "descent with mutations" is a local search metaheuristic, of which the design is very simple and is based on local transformation. It consists in randomly performing random elementary transformations, irrespective improvement or worsening with respect to the objective function. We compare it with another very efficient metaheuristic, which is a simulated annealing method improved by the addition of some ingredients coming from the noising methods. Experiments show that the descent with mutations is at least as efficient for the studied problem as this improved simulated annealing, usually a little better, while it is much easier to design and to tune.

Mathematics Subject Classification. 90C27.

Received April 21, 2017. Accepted June 19, 2018.

1. Introduction

Since the first publication dealing with the noising methods in 1993 [7], several variants of them have been designed and several applications have been studied (see [12] for a survey and references on the noising methods; see more generally [15, 18, 21] or [22] for recent references on metaheuristics, among a rich literature devoted to this topic). In this paper, we deal with a metaheuristic that we may consider as a variant of the noising methods too: the *descent with mutations* (DWM). This method looks like the usual descent, but random elementary transformations are sometimes applied during the resolution, in a blind way, in the sense that they are accepted whatever their effects on the function f to optimize (such an elementary transformation performed without respect to its effect on f will be called a*mutation* in the sequel). The density of performed mutations decreases during the process, so that the process at its end is the same as a classic descent.

Keywords. Metaheuristics, noising methods, simulated annealing, clique partitioning of a graph, aggregation of symmetric relations into median partitions, median equivalence relations.

[1] Telecom ParisTech, 46, rue Barrault, 75634 Paris Cedex 13, France.
[*]Corresponding author: olivier.hudry@telecom-paristech.fr

To experiment this variant, we apply it to two problems, both arising from the field of the aggregation and the approximation of symmetric relations: Régnier's problem [19] and Zahn's problem [25]. They can be represented by weighted graphs; then the aim consists in finding an optimal way to partition the vertex-set of the graph in order to obtain disjoint cliques (see below). The results obtained by the application of DWM to these problems are compared with the ones provided by a hybridization of the simulated annealing method designed in [1] for the same problems with some improving ingredients coming from the noising methods (note that de Amorim *et al.* compared in [1] the applications of tabu search and of simulated annealing to these problems: their experiments show that the results provided by these two metaheuristics are qualitatively quite similar).

In the next section, we detail the principles of DWM. In Section 3, we depict the studied problems (Régnier's problem, Zahn's problem, and the clique partitioning problem) and the chosen elementary transformations allowing us to apply a descent (with or without mutations). The experimental results are described in Section 4. Conclusions can be found in Section 5.

2. Principle of the descent with mutations

As the other metaheuristics, DWM is not designed to be applicable to only one combinatorial problem, but to many of them. Such a problem can be stated as follows:

$$\underset{s \in S}{\text{Minimize}}\, f(s)$$

where S is a finite set and f is a function defined on S; the elements s of S will be called *solutions.*

As many other heuristics, DWM is based on *elementary* (or *local*) *transformations.* A *transformation* is any operation changing a solution into another solution. A transformation will be considered as *elementary* (or *local*) if, when applied to a solution s, it changes one feature of s without modifying its global structure much. For instance, if s is a binary string, a possible elementary transformation would be to change one bit of s into its complement.

Based on these elementary transformations, we may define the *neighbourhood* $N(s)$ of a solution s: $N(s)$ is the set of all the solutions that can be obtained from s by applying an elementary transformation to s. Then, we may define an *iterative improvement method*, or *descent* for a minimization problem (it is the case for the problems considered here), as follows: we start from an initial solution (for instance, randomly generated); for the current solution s, an elementary transformation is applied to s in order to generate a neighbour s' of s; if s' is better than s (*i.e.* $f(s') < f(s)$), then s' becomes the current solution; otherwise we keep s as the current solution; the same process is applied from solutions to solutions, until it is not possible to improve the current solution s^* by means of the elementary transformation:

$$\forall s \in N(s^*), f(s) \geq f(s^*).$$

Then, the descent is over and the final solution s^* provided by the descent is (at least) a *local minimum* of f with respect to the adopted elementary transformation. The whole method may stop here, or may restart a new descent from a new initial solution (to define a *repeated descents method* or *multistart descents*).

In such a descent, the process is not blind in the sense that the elementary transformations are adopted only if they fulfill the following *acceptance criterion*: they must involve an improvement of the value taken by f. In DWM, we also apply the basic process of a descent but, from time to time, instead of applying the previous acceptance criterion, we apply and accept the considered elementary transformation, whatever its effect on f (see Fig. 1, which describes DWM in general): we say that we have a *blind elementary transformation*, or simply a *mutation.* Thus, the only thing to specify in order to apply DWM (in addition to what must be defined to apply a descent, like the elementary transformation) is when a mutation is adopted. It is what we will show in the next section, for the problems studied in this paper.

We said at the beginning that DWM can be considered as a variant of the noising methods. Remember that the most general scheme of the noising methods (see [12]) consists in computing a *noised variation* $\Delta f_{\text{noised}}(s, s')$

```
Repeat:
    - let $s'$ be a neighbour of the current solution $s$
    - with a certain probability, replace the current solution $s$ by $s'$
            (in other words, we apply an arbitrary elementary transformation
             irrespective improvement or worsening: this is a mutation)
    - otherwise, replace the current solution $s$ by $s'$ if we have $f(s') < f(s)$
            (as in a usual descent)
until a given condition is fulfilled.
```

Figure 1. General description of DWM.

of f when a neighbour s' of the current solution s is considered: $\Delta f_{\text{noised}}(s, s') = f(s') - f(s) + \rho$, where ρ is a random number depending on different things (like s, s', the iteration number, the scheme of the noising method, the adopted probability law, and so on); then the acceptance criterion becomes the following: the transformation of s into s' is accepted if $\Delta f_{\text{noised}}(s, s')$ is negative. So, indeed, it is not difficult to design the characteristics of the law followed by ρ in order to show that DWM constitutes a special case of the noising methods: it is sufficient to choose a very negative value for ρ (that is, a negative value with a great absolute value) when we decide to perform a mutation, or 0 otherwise.

On the opposite, we may observe the following main differences with respect to the neighbourhood methods applied for instance in [1], namely the simulated annealing method or the Tabu method, that use a same transformation mechanism:

– in the simulated annealing method, a neighbour s' of the current solution s involving an increase of f, *i.e.* with $f(s') > f(s)$, is accepted with a probability equal to $\exp(-(f(s') - f(s))/T)$, where T is a decreasing parameter called *temperature*; thus the performed transformation is not blind; moreover, the accepting probability requires the computation of an exponential, which is usually time consuming; the tuning of the temperature T (the computation of the initial value of T and the decreasing of T) can be difficult tasks (especially the computation of the initial value of T, even if some procedures can be found in literature), as well as the tuning of the other parameters of this metaheuristic (as the number of transformations applied with a fixed temperature or the criterion to stop the method).
– the difference with Tabu method is still more pronounced since this metaheuristic is not stochastic; remember that the principle of Tabu method is to improve the current solution s as much as possible, when this is possible, or to worsen s as few as possible; in order to avoid to come back to the previous solution when a worsening transformation is performed, this transformation is memorized inside a "Tabu list" and the opposite of this transformation is forbidden during a certain number of iterations; the main difficulty is then to know how long such a transformation should remain tabu; this is related to the length of the Tabu list, which is a parameter that must be tuned.

As shown below by the experiments, a main advantage of DWM is that this metaheuristic is much easier to tune.

3. The clique partitioning problem and the descent with mutations

3.1. Aggregation and approximation of symmetric relations; the clique partitioning problem

The two problems that we consider deal with *the aggregation and the approximation of symmetric relations*, topics that can be found for example in the field of classification and cluster analysis (for a description of the context and for references, see for instance [2,3,5,6,14,20]).

The first one, set by Zahn [25], consists in the approximation of a symmetric relation by an equivalence relation at minimum distance. More precisely, given a symmetric relation R defined on a set X (*i.e.* R is a subset of the Cartesian product $X \times X$), we look for an equivalence relation E (*i.e.* E is a symmetric relation which is transitive), also defined on X, which is at minimum distance to R with respect to the *symmetric difference distance* (this distance is given by the number of pairs $\{x, y\}$ with $(x, y) \in X \times X$ which belong to R but not to E or which belong to E but not to R; it measures the number of disagreements between R and E; see [4]). This problem is NP-hard, as shown by Krivanek and Moravek [17]. Example 3.1 provides an instance of Zahn's problem.

Example 3.1. In Example 3.1, let X be the set $\{a, b, c, d, e\}$ and let R be defined by $R = \{\{a, b\}, \{a, d\}, \{a, e\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}\}$. The equivalence relation E_1 gathering the five elements of X is at distance 3 from R: the disagreements are on the pairs $\{a, c\}$, $\{b, d\}$ and $\{b, e\}$ since these pairs belong to E_1 but not to R. The equivalence relation E_2 gathering$a$, c and e in one class and b and d in another class – such an equivalence will be represented as $ace|\ bd$ – is at distance 5 from R: the disagreements are on the pairs $\{a, b\}$, $\{a, d\}$, $\{d, e\}$ (these pairs belong to R but not to E_2), $\{a, c\}$ and $\{b, d\}$ (these pairs belong to E_2 but not to$R$). An optimal solution (it is not the only one) is $ab \mid cde$, *i.e.* consists in partitioning $\{a, b, c, d, e\}$ into two classes: $\{a, b\}$ and $\{c, d, e\}$, with a number of disagreements equal to 3.

The second problem, set by Régnier [19], consists in the aggregation of equivalence relations into a unique equivalence relation summarizing the initial equivalence relations "as accurately as possible". More precisely, in this problem, we consider a set X of n objects and a set of m criteria; each criterion is assumed to define an equivalence relation on X (or, equivallently, a partition of X); the aim is to find an equivalence relation defined on X which minimizes the total number of disagreements (still provided by the symmetric difference distance) with respect to the given m criteria. This problem is NP-hard when m is not fixed (see [3,16]); when not trivial, its complexity is unknown in the general case when m is fixed. An equivalence relation minimizing this total number of disagreements is called *a median equivalence relation* or, equivalently, *a median partition.*

Example 3.2. In Example 3.2, let X be the set $\{a, b, c, d, e\}$ and consider three ($m = 3$) equivalence relations E_1, E_2 and E_3 defined as follows, with the representation specified in Example 3.1: $E_1 = ade|b|c$ has three classes; $E_2 = ace|bd$ has two classes; $E_3 = acde|b$ has also two classes. Then consider the equivalence relation $E = abe|cd$ with two classes; the number of disagreements betwwen E and E_1 (resp. E_2 and E_3) is equal to 5 (resp. 6 and 6), thus invloving a total number of disagreements equal to 17.

These two problems can be represented (see for instance [23,24] or [16] for the details of the computations) by the following *clique partitioning problem* (CPP in what follows). In CPP, we consider a weighted, undirected and complete graph $G = (X, U, w)$, of which the vertex-set is the set X of objects to gather into the classes of the equivalence relation that we look for. An integer (which can be positive, or negative, or equal to 0) $w(x, y) = w(y, x)$ is associated with each edge $\{x, y\} \in U$ $(x \neq y)$ (see below for the value of this weight in order to reprensent Zahn's problem or Régnier's problem). Then CPP consists in finding a partition of X into $k(G)$ disjoint cliques C_1^*, C_2^*, ..., $C_{k(G)}^*$ in order to minimize the sum of the weights of the edges with their two extremities in a same clique, *i.e.* in order to minimize the function f defined for any partition $(C_1, C_2, \dots, C_k)$

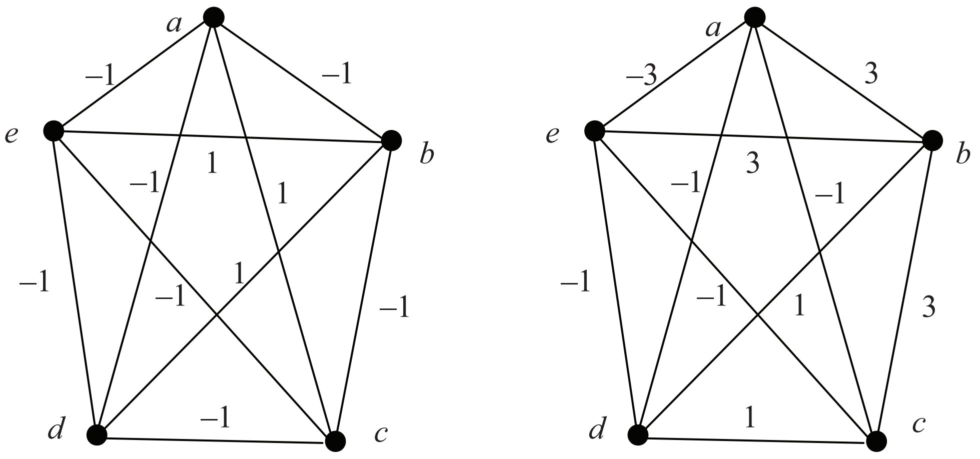

FIGURE 2. The graphs associated with Examples 3.1 and 3.2.

of X by:

$$f(C_1, C_2, ..., C_k) = \sum_{i=1}^{k} \sum_{\substack{\{x,\ y\} \subseteq C_i \\ x \neq y}} w(x, y).$$

Note that the number $k(G)$ of cliques is not fixed a priori and depends on the weighted graph G (for this reason and because of the signs of the weights, CPP is not the famous $k$cut problem, though the formulations of the two problems are close to each other). This CPP is also NP-hard ([23,24]).

To state Régnier's problem and Zahn's problem as instances of CPP, we build a weighted complete graph as follows (see the above references for details). The vertex set will be the set X of Régnier's or Zahn's problems. For Régnier's problem, the weight of an edge $\{x, y\}$ (with $x \neq y$) is given by the difference $m - 2m_{xy}$, where m_{xy} denotes the number of criteria for which x and y are together in a same class (this weight is also the difference between the number, equal to $m - m_{xy}$, of criteria for which x and y are not together in a same class, and the number, equal to m_{xy}, of criteria for which x and y are together); observe that this weight is between $-m$ (if all the criteria gather x and y) and m (if x and y never belong to a same class), with the possibility to be equal to 0 (this requires m to be even). For Zahn's problem, let R be the symmetric relation of the instance; the weight of an edge $\{x, y\}$ (with $x \neq y$) is -1 if x and y are in relation with respect to R and $+1$ otherwise. Then the search of an equivalence relation which is a solution of Régnier's or Zahn's problems consists in both cases in partitioning the weighted complete graph into disjoint cliques in order to minimize the sum of the weights of the edges with their two extremities in a same clique; hence our clique partitioning problem.

Example 3.3. The graphs of Figure 2 are the ones associated with the data of Example 3.1 (on the left) or of Example 3.2 (on the right). For Zahn's problem (on the left), an optimal solution is provided by the 2-class equivalence relation $ab|cde$, with a value for the objective function equal to –4 (another optimal solution is $abcde$, which consists in gathering all the vertices into only one class, or $acde|b...$). For Régnier's problem (on the right), $acde|b$ is the only optimal solution, with a value for the objective function equal to –6.

It is easy to associate an instance of Zahn's problem (a symmetric relation) with any instance of CPP in which all the weights belong to $\{-1, 1\}$. On the other hand, it is shown in [16] how to associate an instance of Régnier's problem (a collection of equivalence relations) with any instance of CPP if all the edge-weights have the same parity. As it is always possible to double the weigths $w(x, y)$ of the edges $\{x, y\}$ without changing the structure of an optimal solution, it appears that it is equivalent to solve Régnier's problem or to solve CPP. And similarly, it is equivalent to solve Zahn's problem or the instances of CPP in which all the weights belong to $\{-1, 1\}$. In the following, we refer to Régnier's problem for instances of CPP with any integer weights, and to Zahn's problem for instances of CPP in which all the weights belong to $\{-1, 1\}$.

3.2. Application of the descent (without mutations) to the clique partitioning problem

To define a descent for CPP, we apply the following elementary transformation (defined by Régnier [19]): we choose a vertex v and we move v from its current clique into another clique or alone in a new clique; in this last case, we say that we put v in the *empty clique*.

Example 3.4. Consider once again the data of Example 3.1 (see also the first graph of Fig. 2). Assume that the current partition is $abc|de$, consisting in gathering a, b and c in a first clique and d and e in a second clique; the value of this partition is equal to $(-1 + 1 - 1) + (-1) = -2$. Then nine elementary transformations could be performed, providing nine neighbours:

- by moving$a$ or b or $c$together with d and e (three transformations); we respectively obtain the equivalence relations $bc|ade, ac|bde, ab|cde$;
- by moving a or b or c alone (three transformations), which creates a new clique; we respectively obtain the equivalence relations $a|bc|de, ac|b|de, ab|c|de$;
- by moving$d$ or $e$together with a, b and c (two transformations); we respectively obtain the equivalence relations $abcd|e, abce|d$;
- by moving d or e alone (only one transformation), which creates a new clique; we obtain the equivalence relation $abc|d|e$.

To apply a descent, we begin from a randomly chosen partition and we consider the vertices one after the other in a cyclic way: the neighbours are ranked in a certain order (which is not necessarily always the same) and they are all considered in this order once, before being considered for a second time; a neighbour better than the current solution is accepted as soon as it has been discovered (see [13] for this way of exploring the neighbourhood). More precisely, for each vertex v and for each clique C (including the empty one and the current clique of v) of the current solution s, we compute the variation $\Delta f(s, C)$ of f when v is moved from its current clique to C; the clique $C^*$for which the variation $\Delta f(s, C^*)$ is minimum is called the *best clique* for$v$ (with respect to s). Note that, since the current clique of v is taken into account, we have $\Delta f(s, C^*) \leq 0$, with an equality if C^* is the current clique of v. Then, if moving v from its current clique to its best clique (with respect to s) involves a strict improvement (that is, if we have $\Delta f(s, C^*) < 0$), we do move v from its current clique to its best clique (with respect to s) and thus we obtain a new solution s' from which we apply the same process, starting from the vertex following v in the prescribed order; otherwise, we keep v in its current clique and we consider the next vertex. When all the vertices are successively considered while there is no vertex that can be moved towards a clique better than its current clique, the descent is over.

3.3. Application of the descent with mutations to the clique partitioning problem

In DWM, we apply almost a descent but, from time to time, instead of moving the considered vertex to its best clique, we move it to a clique chosen randomly. More precisely, when a vertex v is considered, we have two possibilities: with a probability p, we choose a clique randomly, with a uniform probability on the cliques (including the empty one) of the current solution; or, with a probability $1 - p$, we compute the best clique for v; in both cases, we move v to the chosen clique.

The method is utterly described by Figure 3, in which the vertices are assumed to be $1, 2, ..., n$. It requires choosing only two parameters: a real number, called *initialRate*, between 0 and 1, and an integer called *totNbCycles*, which gives the total number of performed *cycles*. We call *cycle* the operations that must be performed in order to compute the best clique for each vertex of the graph (then *totNbCycles* is linked to the total number of applied elementary transformations that are performed during the method, but the explicit relation is not simple because

```
Choose a partition randomly: it is the current partition $P$;
$bestPartition \leftarrow P$;
$numCycle \leftarrow 0$;
while $numCycle < totNbCycles$, do:
    $p \leftarrow initialRate \times \frac{totNbCycles - numCycle}{totNbCycles}$;
    $v \leftarrow 1$;
    while $v \leq n$, do
        choose a real number $q$ between 0 and 1 randomly, with a uniform probability;
        if $q < p$, then choose a clique $C^*$ (which can be the empty clique) in $P$
            randomly, with a uniform probability;
        else compute the best (with respect to $P$) clique $C^*$ for $v$;
        update $P$ by moving $v$ to $C^*$;
        if necessary, update $bestPartition$;
        $v \leftarrow v + 1$;
    $numCycle \leftarrow numCycle + 1$;
apply a descent to $P$;
if necessary, update $bestPartition$;
return $bestPartition$ and $f(bestPartition)$.
```

FIGURE 3. The scheme of DWM for the clique partitioning problem.

the number of disjoint cliques is not fixed and so may change during the process). The parameter *initialRate* gives the probability to perform a mutation at the beginning of the whole process. We can see in Figure 3 that the probability $initialRate \times \frac{totNbCycles-numCycle}{totNbCycles}$ of performing a mutation is computed at the beginning of each cycle and decreases arithmetically from one cycle to the next one, down to 0 (it is also possible to apply a geometrical decrease, as in simulated annealing, but then of course not down to 0). Of course, we keep the best solution obtained during the process in memory. To be sure to reach at least a local minimum, we complete the process with a descent.

4. Experiments

For our experiments, we compare the results of DWM with those provided by repeated descents (RD in the following) and by a method (SA in the following) based on the simulated annealing method developped in [1]: the differences with respect to this method are that the neighbourhood is explored in a cyclic way (see above) and that descents (without mutations) are inserted periodically in the process of SA (a more precise description of this method can be found in [8] or [9]; an application of these improvements to the Travelling Salesman Problem can be found in [10]). Note that this kind of simulated annealing appears, from previous experiments (see [8, 9]), to be among the best methods that we tried on the kind of problems studied here and that it is much more efficient for them than a classic simulated annealing.

It is necessary to tune the parameters of DWM and SA (there is nothing to tune for RD).

For SA, we developped an automatically tuned version which only needs one parameter: the CPU time that the user whishes to spend in order to solve his or her instance [11]. This version computes, for any given instance, a good solution and also good values for the parameters of SA, especially the initial temperature. The experiments reported in [11] show that this automatically-computed initial temperature is very near the initial temperature carefully tuned by an "expert" with statistical tools. For each studied instance, we first compute the "advised" initial temperature with the automatically-tuned version and we then use this initial temperature in a more classic version needing two parameters: the initial temperature (thus initialized with the value automatically computed previously) and the total number of iterations (related to the CPU time that the user whishes to spend in order to solve his or her instance). The CPU time of the first stage of this process (*i.e.* the tuning of the parameters of SA) is not taken into account in the results related to SA and reported below (if we wanted to take the CPU time of the tunings into account, the results provided by SA would be obviously worse than the ones reported below).

For DWM, there is only one parameter to tune (though it is always possible to choose a value equal to 1; see below): *initialRate*. We have not yet an automatically tuned version of DWM. So, for each type of problem (*e.g.* for Zahn's problem with 300 vertices; see below), before the comparisons of the three methods on 100 instances, we choose one instance of this type of problem, and we try for *initialRate* the successive values from 0.05 to 0.4 with steps of 0.025; for each value, we try DWM 100 times. Then the value of *initialRate* which gives the best average result among the 100 trials on this instance is kept for the 99 other instances of the same type, without making new experiments for these remaining 99 instances.

The three methods are compared on the same instances and with the same CPU time. We do not report here all the experiments that we did on DWM for the above problems, but only some results obtained for graphs with integer weights randomly chosen inside an interval $[-W, W]$ for some integer W. However, the qualitative results are quite the same for our other experiments. We considered both Régnier's problem (with different values for W) and Zahn's problem.

In Figure 4, we report the results obtained for graphs with 100, 300 or 500 vertices (the number of vertices is specified in Figure 4 by the column "order") and with W equal to 10 for Régnier's problem (we studied also other possibilities; they lead to the same qualitative conclusions). Thus this provides six cases (two kinds of problems and three numbers of vertices for each problem). For each case, we chose randomly 100 instances, so that Figure 4 synthesizes the results upon 600 instances. For each instance, we performed DWM, SA and RD 10 times. The same CPU time was given to each method; the column "time" specifies the CPU time given to each application of the methods. The six averages on the 100 instances of each case and for each one of the three methods performed 10 times are displayed in Figure 4 in order to compare the 18000 results more easily. In the column "score", the first number gives for each type of problem the number of instances for which DWM is strictly better than SA among the 100 considered instances, and the second number gives the number of instances for which DWM provides a result at least as good as the one obtained by SA. The right three columns give, still for each type of problem, the average value of f among the 100 instances (remember that CPP is a minimization problem: so, the lowest, the best).

From these experiments, we may see that DWM gives results which compare favourably to the ones provided by SA (though the relative differences are not high: less than 0.85%; this is due to the fact that SA is already an excellent method), and much better than the ones of RD. Observe that the gap between SA and DWM seems to increase when the size of the instances grows up (see also the evolution of the scores), even if the relative difference remains small.

Figure 5 shows the evolution of the values taken by f during a typical run of DWM, for an instance of Régnier's problem with 300 vertices; the optimal value for this instance seems to be -14319 (according to

Problem	Order	Time	Score	DWM	SA	RD
Zahn	100	3 s	24-79	–458.76	–458.50	–448.49
Zahn	300	10 s	71-73	–2447.09	–2433.77	–2292.88
Zahn	500	100 s	93-93	–5341.00	–5296.51	–4974.47
Régnier	100	3 s	26-86	–2799.00	–2796.77	–2756.63
Régnier	300	10 s	68-69	–14786.67	–14662.85	–13981.71
Régnier	500	100 s	84-85	–32267.53	–32098.78	–30312.58

FIGURE 4. The average results of the three methods.

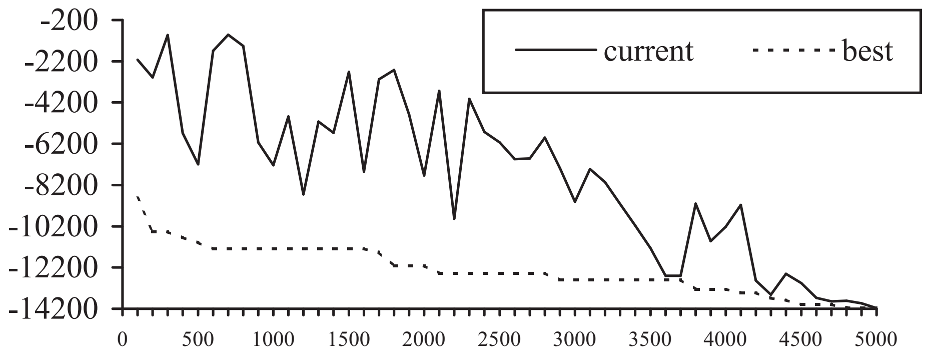

FIGURE 5. Values of f during a run of DWM, at the end of each package of 100 cycles.

extensive experiments with much greater CPU time). More precisely, the horizontal axis corresponds to the number of performed cycles (see Sect. 3.3); here, the total number of cycles is equal to 5000. The vertical axis gives the values taken by f for the current solution (continuous line) and for the best solution computed since the beginning of the run of DWM (dashed line); the reported values in both cases are only the ones obtained at the end of each package of 100 cycles (if we observe the values of f after each performed elementary transformation, the behaviour of the current solution is obviously much more chaotic; in particular, the line associated with the current solution reaches the one associated with the best solution at least when this one changes, and often more frequently...).

In order to get an idea of the sensibility of DWM with respect to *initialRate*, we present in Figure 6 how the average result (still on 100 trials) varies with *initialRate* for the same instance of Régnier's problem as in Figure 5, for a CPU time equal to 10 s. For a same CPU time, the average values computed by SA and by RD are respectively -14069.49 and -13380.37 (they are represented by the dashed lines in Fig. 6). In Figure 6, *initialRate* varies from 0 to 1 with a step equal to 0.025.

From the results of Figure 6, it appears that DWM is not very sensitive to *initialRate* when this parameter is large enough. For values between 0 and 0.1, there are not enough mutations and the behaviour of DWM is worse than the one of SA (though better than the one of RD when *initialRate* is not equal to 0). For values between 0.1 and 1, the behaviour of DWM is quite better and its results compare favourably to the ones of SA, with an optimum around 0.2 or 0.3. The results remain almost the same on the interval [0.2, 0.3] or even on [0.2, 0.5]:

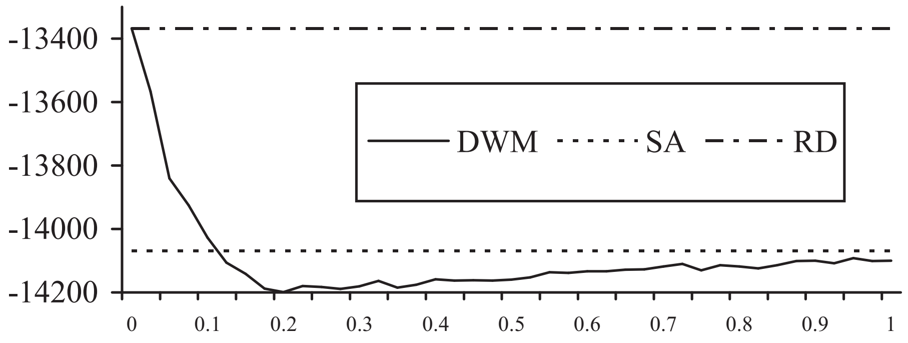

FIGURE 6. Sensibility analysis of DWM to the parameter *initialRate*.

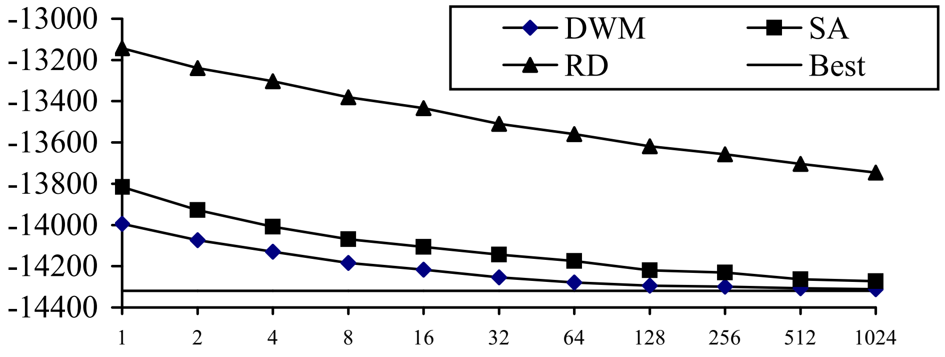

FIGURE 7. Evolution of DWM, SA and RD when the CPU time increases.

the relative difference of the values taken by f is less than 0.14% on [0.2, 0.3] and less than 0.29% on [0.2, 0.5]. After 0.5, when *initialRate* increases to 1, the quality of the results decreases slowly (the values taken by f increase very slowly) but remains good, and reaches the one of the results provided by SA for *initialRate* around 1: the relative difference between the best average computed by DWM and the average value taken by f for *initialRate* $= 1$ is less than 0.7% (the relative difference between the best value computed by DWM and the best value computed by SA is a little more than 0.9%). Of course, inscreasing the CPU time involves an improvement of the performances of DWM. Then the relative difference between the best average computed by DWM and the average value taken by f for *initialRate* $= 1$ still decreases, which means that taking *initialRate* $= 1$ leads also to very good results.

Figures 7 and 8 show how the average (over 100 trials) values of f computed by DWM, SA and RD evolve when the CPU time increases, still for the same instance of Régnier's problem with 300 vertices. More precisely, in both figures, the horizontal axis shows the different CPU times tried in this experiment, from 1 s per trial to 1024 s per trial, with a logarithmic scale.

For Figure 7, the vertical axis gives, for each one of the three methods, the average value of f over 100 trials. The horizontal line above the axis shows what seems to be the optimal value, *i.e.* -14319.

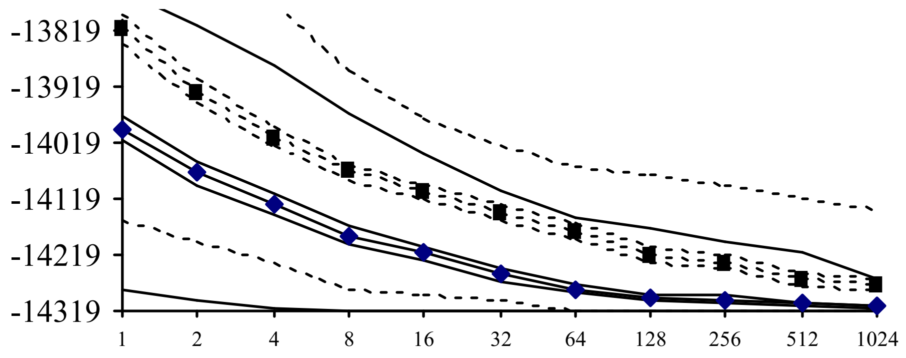

FIGURE 8. Detailed evolution of DWM and SA when the CPU time increases.

Ten lines can be seen in Figure 8, five for DWM (continuous lines) and five for SA (dashed lines). For each method, they provide:

- the evolution of the worst solution computed during the 100 trials, given by the upper (dashed or continuous) line; this line has been troncated for the low values of CPU time;
- the evolution of the best solution computed during the 100 trials, given by the lower (dashed or continuous) line; the line associated with DWM is on the horizontal axis from a CPU time equal to 16 s and just above the axis for CPU times between 4 and 16 s;
- the evolution of the average value over the 100 trials, given by the (dashed or continuous) line with a black diamond (for DWM) or a black square (for SA);
- on both sides of the average line, the two (dashed or continuous) lines giving the 95% confidence interval associated with the considered method. Remember the meaning of this confidence interval. Let X be the random variable associated with the value taken by f for one trial, and let m_0 and σ_0 be respectively the average and the standard-deviation of X. Then it is well-known that, when the number N of trials is large enough (typically, more than 30), the average of the values taken by f during the N trials follows a Gaussian law of parameters m_0 and $\sigma_0/\sqrt{N}$ (this does not depend on the law followed by X). The theoretic average that we would like to know is m_0. Let m and σ be respectively the average and the standard-deviation computed during the N trials; then m_0 can be located in the interval $[m - 1.96\sigma/\sqrt{N},\quad m + 1.96\sigma/\sqrt{N}]$ with a probability equal to 0.95 (that is why this interval is called the 95% confidence interval; this does not mean that 95% of the computed results are in this interval, but that the average value is in this interval with a probability equal to 0.95).

The results displayed on Figures 7 and 8 show that the absolute difference between the average provided by DWM and the one given by SA decreases when the CPU time increases (this is not a surprise if we hope that both methods converge towards an optimal solution, at least quite often), but this difference remains in favour of DWM. Anyway, DWM seems to be more trustable than SA, as shown by the 95% confidence interval, which is tighter for DWM than for SA. Indeed, the standard deviation of DWM is lower than the one of SA. For instance, it is equal to 113 for DWM and to 130 for SA at the beginning of the experiment (with a CPU time equal to 1 s), and it decreases down to 13 for DWM but only to 52 for SA at the end of the experiment (with a CPU time equal to 1024 s). The larger reliability of DWM with respect to SA is also shown by Figure 9, which specifies the number of times that DWM and SA find what seems to be an optimal solution (corresponding to a value of f equal to -14319) during 100 trials, with respect to CPU time (RD never found such a solution in our experiments).

Another way to compare DWM and SA consists in fixing an aimed value of f and to compute the CPU time necessary for each method to reach this goal: then SA needs about 8 times the CPU time consumed by DWM to reach the same value. For instance, within 16 s, the average value reported in Figures 7 and 8 for DWM is

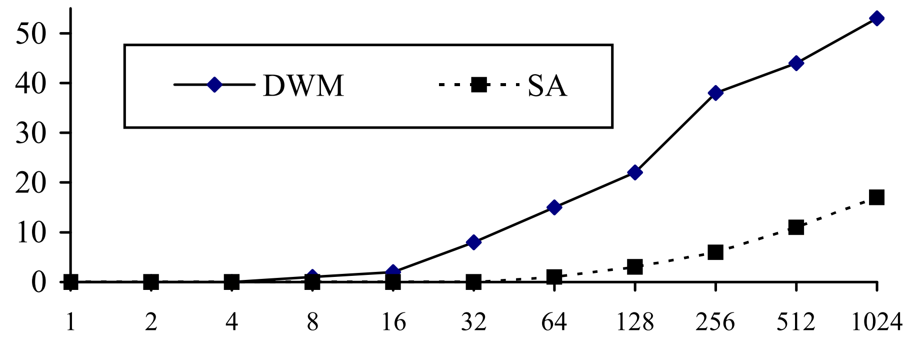

FIGURE 9. Numbers of times that DWM and SA find an optimal solution during 100 trials, with respect to CPU time.

equal to -14219; this value (more precisely, -14216) is reached by SA only after 128 s. The ratio is the same (8) if we consider the first time that what seems to be an optimal solution is found by DWM and by SA. It seems that it increases when the CPU time becomes larger: within 1024 s, SA finds an average value equal to -14272, while DWM finds this value (more precisely, -14279) within only 64 s (thus involving a ratio of 16).

5. Conclusion

From these experiments, it seems that DWM may favourably compete with more sophisticated methods, at least for some combinatorial optimization problems like CPP. It is always possible to say that our way of programming or of tuning SA is not the optimal one and that, otherwise, SA would have given better results. It may be. And it is sure that it is possible to find other tunings and other instances (or other problems!) for which SA would be better than DWM. But it is not the point, or at least not the only one. This study shows experimentally that DWM may provide good results, with about the same (or better) quality than the ones got by an improved version of simulated annealing, within the same CPU time (or less), while it is very easy to design and to apply DWM to problems like CPP, and usually easier to tune than SA.

Indeed, the main advantage of DWM with respect to standard metaheuristics like SA is that there is only one parameter to tune: *initialRate*. The sensibility analysis stated above (the same qualitative behaviour can be observed for other instances of CPP) shows that the tuning of *initialRate* is not a crucial point if the value of *initialRate* is not too low. It is even possible to choose *initialRate* = 1, so that there is no parameter to tune!

It would be interesting to know on which kind of problems DWM can be applied with success, to know whether it depends on the structure of the neigbourhood generated by the elementary transformations or on some combinatorial properties of the studied problems. For instance, the recent application of DWM to a problem of preferences aggregation confirms the previous conclusions. In this problem, we consider individual preferences in an election. These preferences are linear orders defined on a same finite set of candidates. The aim is to determine a linear order which minimizes the total number of disagreements with respect to the individual preferences and which thus can be considered as the collective ranking of the candidates for the issue of the election. Preliminary results [21] show that DWM performs well on this problem. To get a better insight of the behaviour of DWM applied to combinatorial optimization problems will be the topic of future researches.

Acknowledgements. This research was supported by the funds of the "Agence Nationale de la Recherche" as the project "Computation, Communication, Rationality and Incentives in Collective and Cooperative Decision Making" (ANR-14-CE24-0007-01 "CoCoRICo-CoDec"). I would like to thank the anonymous referee for his or her suggestions.

References

[1] S.G. de Amorim, J.-P. Barthélemy and C.C. Ribeiro, Clustering and clique partitioning: simulated annealing and tabu search approaches. *J. Classif.* **9** (1992) 17–42.

[2] P. Arabie, L.J. Hubert and G. de Soete, Clustering and Classification. World Scientific Publishers, Singapore and River Edge, NJ (1996).

[3] J.-P. Barthélemy and B. Leclerc, The median procedure for partitions. *DIMACS Ser. Discrete Math. Theor. Comput. Sci.* **19** (1995) 3–34.

[4] J.-P. Barthélemy and B. Monjardet, The median procedure in cluster analysis and social choice theory. *Math. Soc. Sci.* **1** (1981) 235–267.

[5] G. Brossier, Les éléments fondamentaux de la classification. In: Analyse des données, edited by G. Govaert. Hermès Lavoisier, Paris (2003).

[6] F. Brucker and J.-P. Barthélemy, Éléments de classification. Hermès, Paris (2007).

[7] I. Charon and O. Hudry, The noising method: a new combinatorial optimization method. *Oper. Res. Lett.* **14** (1993) 133–137.

[8] I. Charon and O. Hudry, Mixing different components of metaheuristics. In: Metaheuristics: Theory and Applications, edited by I.H. Osman and J.P. Kelly. Kluwer Academic Publishers, Boston (1996) 589–603.

[9] I. Charon and O. Hudry, Lamarckian genetic algorithms applied to the aggregation of preferences. *Ann. Oper. Res.* **80** (1998) 281–297.

[10] I. Charon and O. Hudry, Application of the noising methods to the Travelling Salesman Problem. *Eur. J. Oper. Res.* **125** (2000) 266–277.

[11] I. Charon and O. Hudry, Self-tuning of the noising methods. *Optimization* **58** (2009) 1–21.

[12] I. Charon and O. Hudry, The noising methods. In: Heuristics: Theory and Applications, edited by P. Siarry. Nova Publishers, New York, NY (2013) 1–30.

[13] K.A. Dowsland, Simulated annealing. In: Modern Heuristic Techniques for Combinatorial Problems, edited by C. Reeves. McGraw-Hill, London (1995) 20–69.

[14] B.S. Everitt, S. Landau, M. Leese and D. Stahl, Cluster Analysis. Wiley, Hoboken, NJ (2011).

[15] M. Gendreau and J.-Y. Potvin, Handbook of Metaheuristics. Springer, Berlin (2010).

[16] O. Hudry, NP-hardness of the computation of a median equivalence relation in classification (Régnier's problem). *Math. Soc. Sci.* **197** (2012) 83–97.

[17] O. Hudry, Descent with mutations applied to the linear order problem. In: Book of Abstracts of the 5th International Symposium on Combinatorial Optimization, *ISCO 2018* (2018) 86–87.

[18] M. Krivanek and J. Moravek, NP-hard problems in hierarchical-tree clustering. *Acta Inf.* **23** (1986) 311–323.

[19] S. Luke, Essentials of Metaheuristics. 2nd edition. Available at: `http://cs.gmu.edu/~sean/book/metaheuristics/`. Lulu Press (2013).

[20] S. Régnier, Sur quelques aspects mathématiques des problèmes de classification automatique. *ICC Bull.* **4** (1965) 175.

[21] C. Romesburg, Cluster Analysis for Researchers. Lulu Press, Morrisville, NC (2004).

[22] P. Siarry (ed.), Metaheuristics. Springer, Berlin (2016).

[23] E.-G. Talbi, Metaheuristics: From Design to Implementation. Wiley, Hoboken, NJ (2009).

[24] Y. Wakabayashi, *Aggregation of binary relations: algorithmic and polyhedral investigations.* Ph.D. thesis, Augsbourg (1986).

[25] Y. Wakabayashi, The complexity of computing medians of relations. *Resenhas* **3** (1998) 323–349.

[26] C.T. Zahn, Approximating symmetric relations by equivalence relations. *SIAM J. Appl. Math.* **12** (1964) 840–847.