On the Influence of the instance structure for metaheuristic performances – Application to a graph drawing problem

Bruno Pinaud, Pascale Kuntz

On the Influence of the Instance Structure on Metaheuristic Performances – Application to a Graph Drawing Problem

Bruno Pinaud and Pascale Kuntz

Abstract— **Metaheuristics are now so common that for some classical hard combinatorial problems, there exist more than ten variants. Thus, the issue of comparing optimization methods is crucial. In this paper, we focus on one aspect of this question: the impact of the choice of the test instances on the metaheuristic performances and the possible link with the fitness landscape structure. We base our experimental framework on the arc crossing minimization problem for layered digraphs. We compare a hybridized genetic algorithm and a multistart descent which are among the best approaches to this problem. We worked on two instance families with various sizes and structural complexities: small graphs which are easy to draw on a standard size support, and large graphs specifically built for our experiments. We show that, for the smallest instances, there is no significant difference between methods whereas for graphs similar to those classically used nowadays in applications the genetic algorithm is better, and for the largest graphs (with a scaling factor up to 10^{300}), the multistart descent is the best method. These results suggest that for "structured" fitness landscapes associated with real-life instances the GA exploits its implicit learning. On the other hand for very large landscapes with probably numerous local optima, only one exploration on a larger scale can be provided by local searches from a random starting point, cheap in computing effort.**

I. Introduction

Solving combinatorial optimization problems with metaheuristics is now so common that for several classical hard problems there can exist more than ten variants ([1], [2]). Despite their stochastic nature, metaheuristics are often a better choice than exact methods. In particular, for medium and large size instances corresponding to nowadays real-life size problems, they allow to find good solutions to a problem with limited computational effort. However, as these approaches become more accessible, the question of their comparison is more and more crucial. According to the literature and our own experience, three key-points have to be carefully taken into account for an efficient comparison: *(i)* the choice of the general objectives of the comparison and the associated measures for both the computational effort and the quality of the solution ([3], [4], [5], [6]), *(ii)* the choice of the test instances ([7], [8], [9]), and *(iii)* the development of a rigorous framework for the experimental tests [10].

Without neglecting the other aspects, we here mainly focus on the influence of the test instances on the metaheuristic performances. Several authors have written guidelines to improve the experimental methodology. But, as far as we know, most authors are more interested in the respective performances of their own heuristics than in a systematic analysis of the influence of the instance structure.

Authors are from the Ecole Polytechnique de l'université de Nantes, Rue Christian Pauc, BP50609, 44306 Nantes cedex 3, FRANCE, {bruno.pinaud, pascale.kuntz}@univ-nantes.fr

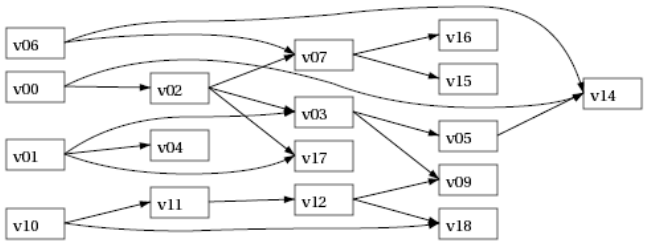


Fig. 1. A layered digraph with 5 vertical layers.

In this paper, for a given problem of graph drawing we precisely show that the instance choice is fundamental when comparing metaheuristics. Generally speaking, this problem, known as the *layered graph drawing problem*, consists in minimizing crossings in a layered layout where the vertices of the graph are arranged on vertical or horizontal layers (Figure 1).

One of the major interests of this problem for our purpose here is that it allows to compare two different strategies: a hybridized genetic algorithm (GA) and a multistart descent (MSD), which are two of the best known approaches to the problem [11]. Moreover, in this case, it is quite easy to define measures of the instance complexity (*e.g.* number of layers, graph density, number of vertices).

We have compared GA and MSD for instances of various structural complexities and various sizes (the ratio is more than 10^{300} between the number of solutions for the smallest and the biggest fitness landscape). We show that according to the instances one method may outperform the other. In particular, our results are consistent with the "No Free Lunch Theorem" ([12], [13]): GA is globally better than MSD for small instances whereas MSD is better than GA for the huge instances specially built for the experiments. Nevertheless, our main purpose here is not a strict comparison of the algorithms nor a generalization of the results obtained to another class of problems; our aim is to contribute to a better understanding of the link between the performance of an optimization process and the fitness landscape (FL) structure.

Completing a previous exhaustive study of small fitness landscapes for this graph drawing problem [14], our results suggest that for "structured" FL associated with small instances, GA exploits its implicit learning. On the other hand for very large landscapes with probably numerous local optima, only one exploration on a larger scale can be

provided by local searches from a random starting point, cheap in computing effort.

The rest of the paper is organized as follows: Section II is a brief review of the choice and construction of the test instances. Section III introduces the notation and the graph drawing problem. Section IV describes both GA and MSD. Section V presents the experimental protocol used. Section VI presents the results. Finally we suggest further paths for future research in Section VII for a better understanding of the difference between both methods.

II. Choice and construction of the test instances

Without a correct design of the test instances and a significant number of instances with a realistic complexity, it is impossible to make a fair and correct comparison between two or more methods. In the design of a representative test set, many factors can have an impact on the performances. For a given factor, it is important to have a sufficiently large variation of its value to correctly measure its influence on the algorithm performance [15]. To our knowledge, one of the most important factors is the size of the instances. The test sets must have an important number of instances corresponding to real-life size problems. Small instances are useful for validating the algorithm and large instances can "stress test" the algorithm and give information on its implementation limits. For a similar purpose, some authors also consider instances which cause the algorithm to fail [7].

The sources of the test instances can also have an impact on the performance. They are varied: real-world data sets, random variants from real data sets, published and online libraries, or random generation of data sets [9]. Each source has its own strengths and weaknesses: real-world data sets are probably the best and thus they have to be used whenever it is possible but they can be hard to obtain (*e.g.* copyright); everybody can use published and online data sets but on the other hand one has to check that these sets cover the whole search landscape; random generation of data set is the quickest and easiest way to obtain data but it is highly controversial, in particular, it is important to check that the variation of the complexity of the generated instances is significant.

In order to simplify the experiments, instances with common properties have to be grouped in classes in order to diminish the variance of the observations [15]. It is not necessary to put instances of the same size in the same class. They just have to be homogeneous and composed of instances with common criteria corresponding to a specific aspect of the problem [3]. Sufficiently large test sets make it possible to resort to statistical tests like the Wilcoxon or Mann-Whitney tests (minimum of about 10 instances) or the t-test (minimum of about 30 instances) for paired samples to assess the significance of the results [4].

III. The layered graph drawing problem

To illustrate the changes in the behavior of the methods when the size and complexity of the instances increase, we base ourselves on a well-known graph drawing problem. Generally speaking, when producing clear and intelligible layouts of a graph, three key points are always considered [16]: the physical constraints inherent to the medium (standard size sheets, computer screen), the drawing conventions (here the layered drawing), and the aesthetics which aims at facilitating readability and memorization of the information embodied in the graph.

From the seminal works of Warfield [17] and Carpano [18] the layered representation has been chosen to highlight directed pairwise relations in numerous systems. The most popular method to draw a layered graph is the "Sugiyama heuristics" introduced by Sugiyama *et al.* [19] and then extended by Eades and Sugiyama [20]. It consists of 4 steps: cycle removal, layer assignment, crossing reduction (by far the most important aesthetic criterion [21]) and finally, vertical coordinate assignment if the layers are represented vertically. Each step is often done with a specific algorithm.

In this paper, we focus on the crossing reduction step. Minimizing crossings in a layered layout could seem intuitively easier than the general problem of minimizing crossings on a plane since the number of crossings is determined by the vertex ordering instead of the vertex geometric coordinates. Yet, it remains *NP*-hard even if there are only two layers [22].

A. Related works

Numerous deterministic heuristics follow the layer-by-layer sweep scheme: vertices of each layer are reordered to reduce crossings while holding the vertex orderings on the other layers. Various strategies have been proposed for reordering. The most commonly used are the sorting methods which exchange vertices using crossing numbers in a way similar to classical sorts [17], and the averaging heuristics with the barycenter heuristics from Sugiyama *et al.* [19], the median heuristics from Eades [23] and their variants ([24], [25]). These heuristics are based on the idea that edge crossings tend to be minimized when connected vertices are placed facing each other. Roughly speaking, these approaches compute the average positions, *i.e.* the barycenter or median of their neighbors, for the vertices on each layer and sort them according to these values.

Different metaheuristics with specific problem-based operators have also been developed: Tabu Search (TS) [26], GRASP [27] or GA ([28], [29], [30], ...). Numerical comparisons reported in [27] show that the TS implementation gives better drawings than the GRASP implementation with a higher computational cost and that both search procedures outperform the deterministic layer-by-layer sweep approaches.

B. Problem formulation and notation

Hereafter we consider an acyclic digraph $G = (V, A)$ with a set V of n vertices, a set A of m arcs, a set $L = \{L_1, L_2, \ldots, L_h\}$ of h layers and a given distribution $V_1, V_2, \ldots, V_h$ of V on L with respectively $n_1, n_2, \ldots, n_h$ vertices. In a layered drawing, every arc $(u, v) \in A$ flows in the same direction: if $u \in L_i$ and $v \in L_j$ then $i < j$.

Algorithm 1 Pseudo-code of the hybridized Genetic Algorithm (GA)

Generate a random *Population* of N genotypes;
While ($\neg$termination criterion) {
  Initialize an empty population $Current\_population$;
  for $i = 0, \ldots, \lceil N/2 - 1 \rceil$ {
    Select two genotypes $g(\Pi)$ and $g(\Pi')$ in $Population$;
    Apply Intra-Layer-Crossover on $g(\Pi)$ and $g(\Pi')$ to create $g(\Pi_i)$ and $g(\Pi'_i)$;
    Apply Inter-Layer-Crossover on $g(\Pi_i)$ and $g(\Pi'_i)$ to create $g(\Pi_c)$ and $g(\Pi'_c)$;
    Apply mutation on $g(\Pi_c)$ and $g(\Pi'_c)$;
    Apply Local Search on $g(\Pi_c)$ and $g(\Pi'_c)$;
    Add $g(\Pi_c)$ and $g(\Pi'_c)$ to $Current\_population$;
    }
  Replace $Population$ by $Current\_population$
}

Moreover, we suppose that the graph is proper *i.e.* each arc $(u, v) \in A$ is connected to vertices on consecutive layers: $u \in L_i$ and $v \in L_{i+1}$. We reach this hypothesis by replacing an arc whose length λ is greater than one by a path of $\lambda - 1$ dummy vertices on consecutive layers.

The vertex ordering on L_k is defined by $\pi_k : L_k \rightarrow \{1, 2, \ldots, |L_k|\}$, where $\pi_k(u) = i$ means that the vertex $u \in L_k$ is on the i^{th} position on L_k and $\sigma_k(i) = \pi_k^{-1}(i)$ indicates the vertex on each position i. A drawing of G is a set of orderings $\Pi = \{\pi_1, \pi_2, \ldots, \pi_h\}$ for each layer.

The problem of minimizing crossings consists in finding an optimal ordering set $\widehat{\Pi}$ so that there is no ordering set with fewer crossings.

IV. Two optimization strategies

In a previous work, we introduced a new GA which, as far as we know, gives better drawings than the other known approaches (especially the TS implementation) [14]. We compared our GA with a multistart descent (MSD) which re-uses some of the GA operators.

A. The hybridized genetic algorithm

GA follows the classical scheme [31] with two specificities: it combines two problem-based crossover operators adapted from ordering GA with a local search strategy based on averaging heuristics (See Algorithm 1).

Each individual $g(\Pi)$ associated with a drawing Π is defined by

$$g(\Pi) = (\sigma_1(1), \ldots, \sigma_1(|L_1|), \sigma_2(1), \ldots, \sigma_2(|L_2|), \sigma_h(1), \ldots, \sigma_h(|L_h|))$$

The selection is determined by a classical roulette wheel based on the arc crossing number as the fitness function. The mutation switches two randomly chosen vertices inside a layer. A generation is here a complete execution of the "while" statement in Algorithm 1; it leads to a renewed population since all parents are replaced by children.

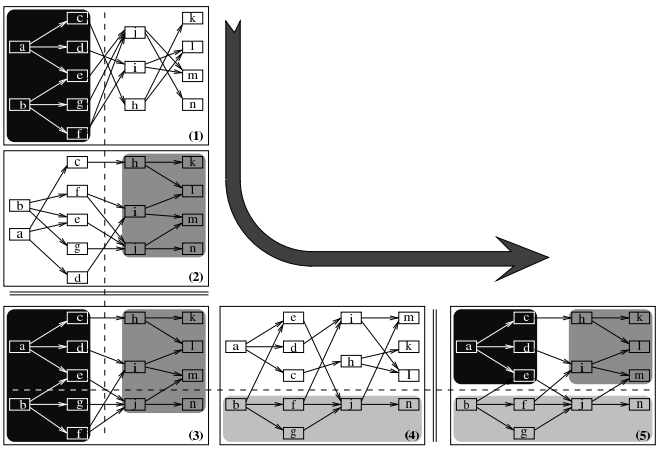


Fig. 2. Improvement of an initial layout with successive sub-graph combinations. The inter-layer combination of (1) and (2) produces (3) and the intra-layer combination of (3) and (4) produces (5) which is a global optimum.

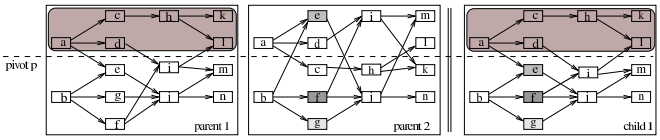


Fig. 3. Example of an application of the intra-layer crossover for the first child only. The part of the graph above the pivot in the first parent is kept as it is. The child is then completed by the missing vertices in their order of appearance in the second parent.

Crossover is inspired from the following observation: the combination of two well-adapted drawings of sub-graphs can produce a better drawing (Figure 2). The combination may be applied between layers (inter-layer crossover) or inside each layer (intra-layer crossover).

The inter-layer crossover is a unique point crossover between layers. The intra-layer crossover aims at combining vertices of a same layer. However, combining blocks inside a layer meets a well-known difficulty for ordinal codings which is to define a crossover which guarantees a feasible solution [32]. The intra-layer crossover is a generalization of the Order Crossover 1 [32] for multi-permutations. A pivot whose random position is normalized by the layer cardinality is defined for each layer of the parents. For the first child, vertex positions of the first parent are retained above the pivot, and below it, positions of missing vertices are completed by those of the second parent according to the vertex ordering (Figure 3). And vice-versa for the second child. Both operators are applied with a given probability.

The hybridization step is a combination of three local operators: greedy switching, adaptive versions of the median and the barycenter heuristics. Each operator is applied sequentially with a given probability on each layer. We have experimentally shown that, for this problem, adding a local search in the classical GA scheme has a great influence on three major characteristics of the optimization process: improving the convergence towards the best solution, saving computation time and reducing the variability of the results

Algorithm 2 Pseudo-code of the multistart descent strategy.

```
do {
  continue ← 0
  Apply the greedy-switch operator to each layer of an individual g(Π) to create g(Π_s);
  if the crossing number decreases {
    Replace g(Π) by g(Π_s);
    continue ← 1;
    }
  Apply the median operator to each layer of g(Π) to create g(Π_m);
  if the crossing number decreases {
    Replace g(Π) by g(Π_m);
    continue ← 1;
    }
  Apply the barycenter operator to each layer of g(Π) to create g(Π_b);
  if the crossing number decreases {
    Replace g(Π) by g(Π_b);
    continue ← 1;
    }
} while continue == 1;
Return g(Π);
```

inherent to the stochastic approaches.

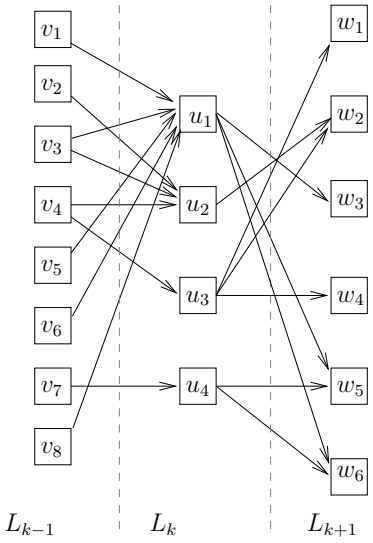
B. The multistart descent

The MSD re-uses the three operators defined for the GA hybridization. Let us describe them more precisely for a layer L_k. The greedy-switch heuristic switches random consecutive vertex pairs. The median and barycenter heuristics compute a new arrangement for a complete layer. The new position of a vertex is a function of its neighbors' position on L_{k-1} and L_{k+1}. First, the median or barycenter position of all the vertices of a layer is computed (see [11] for more details). Then, this set of positions is sorted in increasing order. The new arrangement of the layer is directly deduced from the sorted set of median or barycenter position. The underlying idea is to reduce the length of the arcs by putting the arcs horizontally as much as possible (see Figure 4).

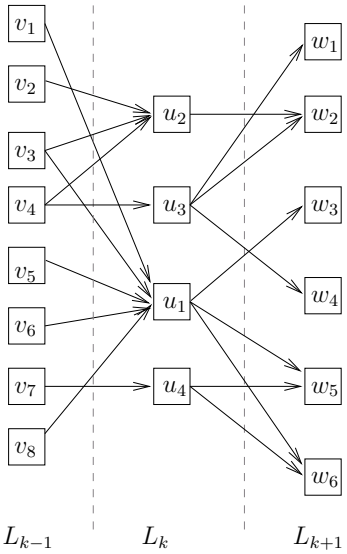
Each of these operators is sequentially applied on each layer L_k for $k = 1, \ldots h$. And following a descent strategy the solution is kept whenever crossings decrease (see Algorithm 2). In order to compare MSD and GA on a same time scale, the stopping time of MSD has been set to the convergence time of the GA best solution.

V. Experimental protocol

Previous experiments described in [11] have shown that the best probabilities for each operator are: 0.02 for mutation, 0.2 for each crossover, 0.05 for the greedy switch operator and 0.2 for median and barycenter. The population size is set to 100 individuals. GA stops when no improvement has occurred after 100 generations. Preliminary experiments have shown that the larger the populations and the more generations before stopping GA, the better results. But this



(a) Original drawing with 27 crossings.



(b) Drawing after permutation of the vertices of layer L_k by their median position. 9 crossings remain.

Fig. 4. Example of an application of the median operator.

choice is a compromise between the quality of the solution and the computational cost. Because of the stochastic nature of the methods, we did 100 runs of each algorithm for each graphs.

GA and MSD are coded with the C programming language and experiments were done on a dual AMD Athlon at 2GHz with Linux 2.6. The code of the local search operators is shared between GA and MSD.

The source for the test instances is a random layered digraph generator similar to the one previously developed for tests with metaheuristics ([26], [27]). Three parameters can vary: the number of layers, the number of vertices per layer and the graph density. A commonly used density measure is the ratio of the arc number m to the arc number

	GA	MSD	GA=MSD
$d(G) = 0.3$	50.55	40.1	9.35
$d(G) = 0.5$	48.77	42.52	8.72
$d(G) = 0.7$	39	47.22	13.78

TABLE I

DISTRIBUTION IN % OF THE BEST COMPUTED SOLUTION FREQUENCY BETWEEN GA AND MSD.THE ROW GA=MSD INDICATES THAT BOTH HEURISTICS FIND A SOLUTION WITH THE SAME CROSSING NUMBER.

h	GA	MSD	GA=MSD
4	44.3	18.1	37.6
8	39.45	58.65	1.9
12	33.25	64.9	1.85

TABLE II

WITH $d(G) = 0.7$, DISTRIBUTIONS IN % OF THE BEST COMPUTED SOLUTION FREQUENCY FOR DIFFERENT LAYER CARDINALITIES.

of a complete graph of the same size, but we prefer a more suitable definition for layered digraphs: as the maximal layered digraph has

$$m_{\max} = \sum_{k=2}^{h} n_{k-1} \times n_k$$

arcs, the layered density is $d(G) = m/m_{\max}$. We use two sets of 180 graphs of various sizes to cover a large spectrum of possibilities. We have a maximum scaling factor of only 13 for the number of vertices and 7 for the number of layers. But, the order of the search landscape, composed of all the possible drawings, is $\prod_{k=1}^{h} n_k!$. Therefore there is a maximum scaling factor of about 10^{314} in the size of the search landscape.

To compare the two methods, we compute the "best solution frequency": for each run of GA and MSD on the same graph we count the number of runs for which only one method has reached the best solution. We additionally consider identity cases where both methods reach a solution with the same crossing number; this information, often neglected in the literature, has here been proved relevant. The results are confirmed by two non-parametric statistical tests: the Wilcoxon signed rank test and the *t-test* (or student test) for paired samples on the mean crossings number for a standard level of significance $\alpha = 0.05$.

VI. COMPUTATIONAL EXPERIMENTS

We have compared the respective performances of GA and MSD on two graph families: one composed of small graphs easy to draw on a standard size support and the other composed of large graphs built especially for the comparison.

A. Experiment on small graphs

The generator was used to create 20 instances for each combination of 4, 8 and 12 layers with $d(G) = 0.3$, 0.5 and 0.7 (total of 180 graphs). The vertex number per layer is randomly chosen between 5 and 15. These sizes are well-representative of small graphs with intelligible layouts on a standard size sheet or a single computer screen. The vertices are often represented by labeled boxes and can rarely exceed 70 or 80 on a standard size sheet ([33], [16]).

Table I shows the results of the comparison between GA and MSD. For the densities $d(G) = 0.3$ and $d(G) = 0.5$, GA outperforms MSD. These results are confirmed by the statistical tests: for $d(G) = 0.3$ (resp. $d(G) = 0.5$) the p-value of the Wilcoxon test is equal to 1.712×10^{-4} (resp. 5.652×10^{-10}) and the p-value of the t-test is equal to 1.152×10^{-4} (resp. 1.597×10^{-10}).

However, for $d(G) = 0.7$ the situation is more confused: MSD more often finds better solutions when we only take into account strict comparisons, but in almost 14% of the cases the two approaches are equivalent. When studying more precisely the behavior of the two approaches for different layer cardinalities, it appears that the equivalent cases are concentrated on graphs with a small number of layers (Table II). For $d(G) = 0.7$ and $h = 4$ the p-value of the Wilcoxon test is equal to 0.15; this result confirms the difficulty to separate the two approaches in this case, which corresponds to relatively small fitness landscapes. For graphs of the same density ($d(G) = 0.7$) with a higher number of layers, MSD outperforms GA. We believe that this change in the respective behaviors of the two metaheuristics is associated with an important modification of the structure of the fitness landscape with probably a significant increasing of the number of local optima. To confirm this trend, we have investigated their behaviors for very complex graphs.

B. Experiment on large graphs

The generator was used to create 20 instances for each combination of 20, 25 and 30 layers for $d(G) = 0.6$, $d(G) = 0.65$, $d(G) = 0.7$. The vertex number per layer is randomly chosen between 10 and 35. For practical applications, we are aware that layered layouts are not adapted for this kind of graphs. Here, the objective is to compare the performance of GA and MSD in their abilities to explore a very complex fitness landscape. To assure a certain convergence of the GA, we stop the algorithm when no improvement has arisen during 500 consecutive generations. Due to the high complexity of the considered cases, the computational effort was very important: 8 months were necessary to compute the 18,000 runs of the two metaheuristics.

The result is self evident (Table III): MSD always finds better results than GA excepts for 2 runs over the 18,000 runs.

Completing a previous analysis on the fitness landscape structure associated with the local search operators [14] leads us to conclude that for this graph class, there are numerous local optima similar in value. Their distribution in the fitness landscape does not allow to fully take advantage of the efficiency of the GA implicit learning. In this case, local searches from random starting points, cheap in computing effort, provide an exploration on a large scale.

h	GA	MSD	GA=MSD
20	0	100	0
25	0	100	0
30	0	100	0

a. $d(G) = 0.6$

h	GA	MSD	GA=MSD
20	0.05	99.95	0
25	0	100	0
30	0	100	0

b. $d(G) = 0.65$

h	GA	MSD	GA=MSD
20	0.05	99.95	0
25	0	100	0
30	0	100	0

c. $d(G) = 0.7$

TABLE III

DISTRIBUTIONS IN % OF THE BEST COMPUTED SOLUTION FREQUENCY FOR THE HUGE GRAPH SET.

VII. CONCLUSION

Comparing and choosing metaheuristics for a given problem is perhaps one of the most difficult task in evolutionary computation which must take into account different factors. In this paper, we have focused on the influence of the choice of the test instances and the link with the metaheuristic performances. We have analyzed a large number of instances, from very small and simple ones to very large and complex ones especially built for the behavioral analysis and far from practical applications. We have showed that depending on the size and complexity of the instance, the best method is not always the same: for small and easy instances there is no significant difference between methods, for larger instances compatible with real-life problems, the GA is the best method and for the most complex graphs, MSD is the best method. This change in the result may be due to an important variation in the fitness landscape structure and the presence of numerous local optima.

This underlines the importance of developing methods that can leave the basin of attraction of a local optima as late as possible in the search process. The fitness landscape analysis [14], based on graph modeling initially proposed by Jones and Forrest [34] has to be extended to larger graphs in order to better understand the variation of metaheuristic behaviors [35]. Further works could associate both combinatorial approaches from graph theory [36] and statistical approaches ([37], [38]) which aim at defining correlation measures so as to quantitatively evaluate the landscape ruggedness.

ACKNOWLEDGMENT

The authors would like to thank Ms. Catherine Galais for her help and the anonymous reviewers for their enlightening comments.

REFERENCES

[1] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[2] L. Bianchi, M. Dorigo, L. Gambardella, and W. Gutjahr, "Metaheuristics in stochastic combinatorial optimization: a survey," Tech. Rep. IDSIA-08-06, IDSIA, Dalle Molle Institute for Artificial Intelligence, Switzerland, 2006.

[3] A. Eiben and M. Jelasity, "A critical note on experimental research methodology in EC," in *Proc. of the 2002 Congress on EC (CEC'2002)*, pp. 585–587, IEEE Publications, 2002.

[4] E. Taillard, "A statistical test for comparing success rates," in *Proc.of the* 5^{th} *Metaheuristics Int. Conf*, 2003.

[5] M. Birattari and M. Dorigo, "How to assess and report the performance of a stochastic algorithm on a benchmark problem: *mean* or *best* result on a number of runs?," *Optimization letters*, 2007. To appear.

[6] D. Shilane, J. Martikainen, S. Dudoit, and S. Ovaska, "A general framework for statistical performance comparison of evolutionary computation algorithms," in *Proc. of the* 24^{th} *IASTED Conf. on Artificial Intelligence and Applications*, pp. 7–12, ACTA Press, 2006.

[7] R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart, "Designing and reporting on computational experiments with heuristic methods," *J. of Heuristics*, vol. 1, no. 1, pp. 9–32, 1995.

[8] M. Coffin and M. Saltzman, "Statistical analysis of computational tests of algorithms and heuristics," *INFORMS J. on Computing*, vol. 12, no. 1, pp. 24–44, 2000.

[9] R. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *J. of Heuristics*, vol. 7, no. 3, pp. 261–304, 2001.

[10] J. Hooker, "Testing heuristics: We have it all wrong," *J. of Heuristics*, vol. 1, no. 1, pp. 33–42, 1995.

[11] P. Kuntz, B. Pinaud, and R. Lehn, "Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm," *J. of Heuristics*, vol. 12, no. 1–2, pp. 23–36, 2006.

[12] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[13] Y. C. Ho and D. L. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *J. of Optimization Theory and Applications*, vol. 115, no. 3, pp. 549–570, 2002.

[14] P. Kuntz, B. Pinaud, and R. Lehn, "Elements for the description of fitness landscapes associated with local operators for layered drawings of directed graphs," in *Metaheuristics: Computer Decision-Making* (M. Resende and J. de Sousa, eds.), vol. 86 of *Applied optimization*, pp. 405–420, Kluwer Academic Publishers, 2004.

[15] E. Taillard, "Few guidelines for analyzing methods," in *Tutorial,* 6^{th} *Metaheuristics Int. Conf.*, 2005.

[16] G. Di-Battista, P. Eades, R. Tamassia, and I.-G. Tollis, *Graph drawing – Algorithms for the visualization of graphs*. Prentice-Hall, 1999.

[17] J. Warfield, "Crossing theory and hierarchy mapping," *IEEE J. on Systems, Man and Cybernetics*, vol. 7, no. 7, pp. 505–523, 1977.

[18] M. Carpano, "Automatic display of hierarchized graphs for computer aided decision analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. 10, no. 11, pp. 705–715, 1980.

[19] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, no. 2, pp. 109–125, 1981.

[20] P. Eades and K. Sugiyama, "How to draw a directed graph," *Journal of Information Processing*, vol. 4, no. 13, pp. 424–437, 1990.

[21] H. Purchase, "Effective information visualisation: a study of graph drawing aesthetics and algorithms," *Interacting with computers*, vol. 13, no. 2, pp. 147–162, 2000.

[22] M. Garey and D. Johnson, "Crossing number is NP-complete," *J. Algebraic Discrete Methods*, vol. 4, no. 3, pp. 312–316, 1983.

[23] P. Eades and N. Wormald, "Edge crossings in drawings of bipartite graphs," *Algorithmica*, vol. 11, pp. 379–403, 1994.

[24] E. Gansner, S. North, and K. Vo, "A program that draws directed graphs," *Software - Practice and Experience*, vol. 18, no. 11, pp. 1047–1062, 1988.

[25] L. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan, "A browser for directed graphs," *Software - Practice and Experience*, vol. 17, no. 1, pp. 61–76, 1987.

[26] M. Laguna, R. Marti, and V. Valls, "Arc crossing minimization in hierarchical design with Tabu Search," *Computers and Operations Res.*, vol. 24, no. 12, pp. 1175–1186, 1997.

[27] R. Marti, "Arc crossing minimization in graphs with GRASP," *IIE Trans.*, vol. 33, no. 10, pp. 913–919, 2001.

[28] E. Makinen and M. Sieranta, "Genetic algorithms for drawing bipartite graphs," *Int. J. of Comput. Math.*, vol. 53, no. 3, pp. 157–166, 1994.

[29] A. G. Tettamanzi, "Drawing graphs with evolutionay algorithms," in *Proc. of Conf. on Adaptative Computing in Design and Manufacture, ACDM*, pp. 325–338, 1998.

[30] J. Utech, J. Branke, H. Schmeck, and P. Eades, "An evolutionary algorithm for drawing directed graphs," in *Proc. of the Int. Conf. on Imaging Science, Systems and Technology*, pp. 154–160, CSREA Press, 1998.

[31] D.-E. Goldberg, *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[32] D. Whitley and N. Yoo, "Modeling simple genetic algorithms for permutation problems," in *Fundations of Genetic Algorithms III*, Morgan Kaufmann, 1995.

[33] I. Herman, G. Melançon, and M. Marshall, "Graph visualization and navigation in information visualization: a survey," *IEEE Trans. on Visualization and Computer Graphics*, vol. 6, no. 11, pp. 24–43, 2000.

[34] T. Jones, *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Alburquerque, 1995.

[35] C. Fonlupt, D. Robilliard, P. Preux, and E. Talbi, *Advances and Trends in Local Serach Paradigms for Optimization*, ch. Fitness Landscapes and Performance of Metaheuristics, pp. 345–358. Kluwer Academic Press, 1999.

[36] C. Reidys and P. Stadler, "Combinatorial landscapes," *SIAM Review*, vol. 44, pp. 3–54, 2002.

[37] L. Kallel, B. Naudts, and C. Reeve, "Properties of fitness functions and search landscapes," in *Theoretical Aspects of Evolutionary computing* (L. Kallel, B. Naudts, and A. Rogers, eds.), pp. 175–206, Springer, 2001.

[38] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in *Proc. of the Sixth Conf. on Genetic Algorithms*, pp. 184–192, Morgan Kaufmann, 1995.