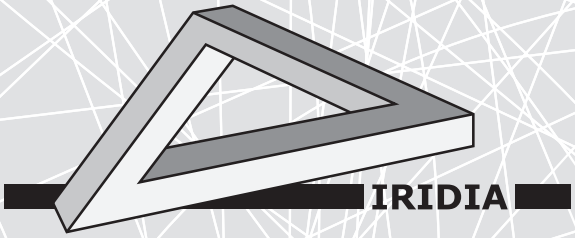


**Université Libre de Bruxelles**

*Institut de Recherches Interdisciplinaires*
*et de Développements en Intelligence Artificielle*

# Hierarchical Iterated Local Search for the Quadratic Assignment Problem

Mohamed Saifullah Hussin and Thomas Stützle

**IRIDIA – Technical Report Series**

Technical Report No.
TR/IRIDIA/2009-021

June 2009

**IRIDIA – Technical Report Series**
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2009-021

# Hierarchical Iterated Local Search for the Quadratic Assignment Problem

Mohamed Saifullah Hussin and Thomas Stützle

June 2009

**Abstract**

Iterated local search is a stochastic local search (SLS) method that combines a perturbation step with an embedded local search algorithm. In this article, we propose a new way of hybridizing iterated local search. It consists in using an iterated local search as the embedded local search algorithm inside another iterated local search. This nesting of local searches and iterated local searches can be further iterated, leading to a hierarchy of iterated local searches. In this paper, we experimentally examine this idea applying it to the quadratic assignment problem. Experimental results on large, structured instances show that the hierarchical iterated local search can offer advantages over using a "flat" iterated local search and make it a promising technique to be further considered for other applications.

## 1 Introduction

Hybrid stochastic local search (SLS) algorithm are those that combine various other SLS strategies into one more complex SLS algorithm [9]. In this sense, iterated local search (ILS) is such a hybrid SLS method since it combines local search (one strategy) with perturbations (another strategy) that are typically of a different nature than the modifications applied in the local search [10]. Following another nomenclature, ILS is also classified as being one (typically non-hybrid) metaheuristic. One goal of the research area of *hybrid metaheuristics* is to study high-level strategies that combine *different* metaheuristics into a new metaheuristic [3]. If, hence, the goal is to propose a hybrid metaheuristic, in the case of ILS this is a rather trivial undertaking. If we use inside ILS instead of an iterative improvement local search a tabu search or a simulated annealing, two examples of other metaheuristics, we already have a hybrid metaheuristic. In fact, such proposals have studied [15, 5, 12, 13]. Recently, also hybrids between ILS and evolutionary algorithms have been considered, where the evolutionary algorithm plays the role of the perturbation operator in ILS [11].

In this article, we propose a new way of generating a hybrid "metaheuristic": we hybridize a metaheuristic with itself! To be more concrete, here we propose to hybridize ILS with ILS. In fact, this is can be accomplished by using inside one ILS algorithm a local search procedure that is another ILS algorithm. Such a type of hybridization can actually make sense. In fact, many hybrid algorithms are obtained by using some higher level "perturbation mechanism" and combining this with a local search type algorithm. This is the case, for example, for hybrid algorithms that combine evolutionary algorithms with local search algorithms (resulting in so called memetic algorithms) or that combine ant colony optimization with local search algorithms. Note that in these cases, the local search part is usually taken by iterative improvement algorithms but also many examples exist of using tabu search, simulated annealing or even iterated local search [7]. In our proposed ILS-ILS hybrid, the outer ILS algorithm plays this role of the perturbation mechanism while the inner ILS algorithm plays the role of the local search.

Interestingly, this idea of nesting iterated local searches can be further iterated, by hybridizing ILS with an ILS-ILS hybrid and so on. Hence, we can define a hierarchy of nested ILS algorithms. Given this possibility, we have denoted this class of hybrids as hierarchical ILS (HILS). In fact,

**Algorithm 1** Iterated Local Search

**procedure** *ILS($s_0$)*
**input**: $s_0$ % initial solution
$s^* = Local\_Search(s_0)$
**while** termination condition is not met **do**
  $s' = Perturbation(s^*, history)$
  $s^{*\prime} = Local\_Search(s')$
  $s^* = Acceptance\_Criterion(s^*, s^{*\prime}, history)$
**end while**
**return** $s^*_b$ % best solution found in the search process
**end** *Iterated Local Search*

the ILS-ILS hybrid is simply the first level of this hierarchy with one ILS on top and one in the next lower level; therefore, we denote this hybrid as HILS(2); HILS(1) is a flat ILS without any nested ILS process.

In this paper, we explore this idea of hierarchical ILS applying it to the quadratic assignment problem (QAP). In particular, we build an HILS(2) algorithm using two particular ILS algorithms as the underlying building blocks. The computational results obtained with the resulting HILS(2) algorithm are very promising. On a number of instances, a tuned version of HILS(2) performs significantly better than tuned versions of the underlying ILS algorithms. Some limited experiments with an untuned HILS(3) algorithm show that the consideration of further levels of this hierarchy may be interesting for QAP instances with a particular structure.

This article is structured as follows. In the next section, we give an overview of ILS, introduce the hierarchical ILS approach in more detail, and illustrate in Section 3 its adaptation to the QAP. In Section 4 we introduce the benchmark instances used in this article. Experimental results with the hierarchical ILS are given in Section 5 and we conclude in Section 6.

## 2 Hierarchical iterated local search

Iterated Local Search (ILS) iterates over a heuristic, in the following called *local search*, by building a chain of local optima. It builds this chain by perturbing at each iteration a current local optimum; this results in an intermediate solution, which is the starting solution of the next local search. Once the local search terminates, the new local optimum is accepted depending on some acceptance test. An algorithmic outline of ILS is given in Algorithm 1. ILS starts building this chain of solutions using some initial solution and applying a local search to it. The goal of the perturbation is to modify a current local optimum to generate a good new starting solution for a local search. The perturbation should not be too strong, since otherwise the perturbation is close to a random restart; but also not too weak, since otherwise the following local search could simply undo the changes and return to the previous local optimum. The acceptance criterion decisively determines how greedy the search is. For a more detailed discussion of ILS we refer to [10].

The role of the *Local_Search* procedure can be played by any heuristic that takes as input some solutions $s \in S$, where $S$ is the space of all candidate solutions for the problem under concern, and outputs an improved solution $s^*$. In fact, for the following discussion only this input–output behavior is important, that is, we can see local search as a function $Local\_Search : S \mapsto S^*$; we call $S^* \subseteq S$, for simplicity, the space of "local optima". If the local search is an iterative improvement algorithm, $S^*$ is the space of local optima w.r.t. the neighborhood structure used in the local search; if the local search is a tabu search, then $S^*$ is the set of the best solutions found by a tabu search execution of a specific search length and using a specific $s$ as input. This point of view results in the interpretation of ILS as being a stochastic search in the space of local optima, the space of "local optima" being defined by the output behavior of the embedded heuristic.

The underlying idea of the hierarchical ILS is to use two (or more) nested iterated local searches. The first level of this hierarchy is obtained by replacing the procedure *Local_Search* in Algorithm 1

**Algorithm 2** Hierarchical Iterated Local Search; level 2

**procedure** *Iterated Local Search*
$s_0 =$ *Generate_Initial_Solution*
$s^{**} = ILS(s_0)$
**while** termination condition is not met **do**
  $s' =$ *Perturbation*$(s^{**}, history)$
  $s^{**\prime} = ILS(s')$
  $s^{**} =$ *Acceptance_Criterion*$(s^{**}, s^{**\prime}, history)$
**end while**
**return** $s_b^{**}$, the best solution found by HILS
**end** *Iterated Local Search*

by an ILS algorithm. This results in an HILS(2) that is outlined in Algorithm 2. This outline corresponds to the first hierarchy level of HILS and it will be denoted by HILS(2), since it consists of two nested ILS algorithms.

When applying the above interpretation of ILS as searching the space of local optima of *Local_Search* to HILS(2), we get the following relationship. The procedure *ILS* can be interpreted as a function $ILS : S^* \mapsto S^{**}$, that is, it is simply defining a mapping from the space of local optima $S^*$ to a smaller space $S^{**}$. We have that $S^{**} \subseteq S^*$ and it can be interpreted as the set of possible outputs of the inner ILS, that is, it is the set of local optima that correspond to the potential best solutions that are found by the inner ILS. The "outer" ILS is, hence, doing a stochastic search in a potentially even smaller space than $S^*$. Clearly, this hierarchy could be further extended by considering further levels, leading to an interpretation of HILS as searching in $S^*$, $S^{**}$, $S^{***}$, and so on, depending on the number of nested iterated local searches. In this article, we examine in some more detail an HILS(2) algorithm; in addition, we provide some evidence that at least for some class of QAPLIB instances an HILS(3) algorithm shows very good performance.

# 3 Hierarchical ILS for the quadratic assignment problem

## 3.1 Quadratic assignment problem

The QAP is an NP-hard problem [14] that has attracted a large number of research efforts [2, 4]. It is an abstract model of many practical problems arising in applications such as hospital, factory or keyboard layout. In the QAP, one is given two $n \times n$ matrices $A$ and $B$, where $a_{ij}$ gives the distance between the pair of positions $i$ and $j$; $b_{kl}$ gives the flow of, for example, patients or material between units $k$ and $l$. Assigning units $k$ and $l$ to positions $i$ and $j$, respectively, incurs a cost contribution of $a_{ij} \cdot b_{kl}$. The goal of the QAP is to assign units to positions such that the sum of all cost contributions is minimized. A candidate solution for the QAP can be represented as a permutation $\pi$, where $\pi(i)$ is the unit that is assigned to position $i$. The objective function of the QAP is

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi(i)\pi(j)}. \quad (1)$$

Due to its difficulty for exact algorithms, currently SLS algorithms define the state-of-the-art for the QAP [9]. ILS algorithms have shown very good performance on the QAP, some variants reaching state-of-the-art performance [16].

## 3.2 Iterated local search for the quadratic assignment problem

Various ILS algorithms for the QAP have been studied in [16] and we describe here the main ILS components taken from [16] for our study.

Our ILS and HILS algorithms start from a randomly generated initial solution. As the local search we consider an iterative improvement algorithm in the two-exchange neighborhood using a first improvement pivoting rule. The two-exchange neighborhood $\mathcal{N}(\pi)$ of a solution $\pi$ is given by the set of permutations that can be obtained by exchanging units $r$ and $s$ at positions $\pi_r$ and $\pi_s$. As soon as an improving move is found during the exchange, it is immediately applied. To avoid spending too much time during the search, the iterative improvement algorithm exploits the *don't look bits* technique to speed up the local search algorithm. If no improvement is found during the neighborhood scan of an unit, the corresponding *don't look bit* is turned on (that is, set to one) and the unit is not considered as a starting unit for the next iteration. If an unit is involved in a move and changes its position, its *don't look bit* is turned off again (that is, set to zero). The purpose of using *don't look bits* is to restrict the attention to the most interesting part of the local search.

The perturbation in ILS and HILS exchanges $k$ randomly chosen units. We consider various possibilities for defining the strength of the perturbation, that is, the number $k$ of units exchanged and consider two specific schemes: keeping the number $k$ fixed or using a variable perturbation strength, as done in Variable Neighborhood Search (VNS) [8].

The main differences among the ILS algorithms studied by Stützle [16] concerned the acceptance criteria. In particular, he studied the better acceptance criterion (*Better*), where a new local optimum $s^{*\prime}$ is accepted if it is better than the incumbent one $s^*$, a random walk acceptance criterion (*RandomWalk*), where a new local optimum $s^{*\prime}$ is accepted irrespective of its quality (that is, always), and a restart acceptance criterion (*Restart*), which behaves like *Better* with one exception: if for $i_r$ successive iterations no improved solution has been found, the ILS algorithm is restarted from a new, randomly generated solution.

Note that in [16] an ILS algorithm based on a Simulated Annealing type acceptance criterion (*LSMC*) was also studied. We did not use this one here, mainly to simplify the construction of an appropriate HILS algorithm by needing to adapt less parameters. Another reason is that for most QAP instances, the best results of the ILS algorithms that use the *Better*, *RandomWalk*, and *Restart* acceptance criteria are roughly on par or better than those of *LSMC* [16]. Similarly, the two population-based ILS variants replace worst (*RepWorst*) and evolution strategies (*ES*) are not considered as building blocks here, since the target here is an ILS variant that does not use a population.

### 3.3 Hierarchical iterated local search

For defining an HILS(2) algorithm, we need to define the acceptance criteria and perturbations used at each of the hierarchy levels—the local search algorithm is only used in the inner ILS.

The construction of the HILS(2) algorithm is based on the following main rationale. The inner ILS should provide a fast search for good solutions in a confined area of the search space, while the outer ILS loop is responsible for guiding the search towards appropriate regions of the search space. We translate this into the outer ILS algorithm doing rather large jumps in the search space, corresponding to large perturbations, while the inner ILS algorithm is only using small perturbations.

The only remaining component to be set is the acceptance criterion. In this article, we consider for both hierarchy levels only two possibilities: either using *Better* or *RandomWalk*. This results in a total of four possible combinations. We compare the HILS(2) algorithm also to ILS with *Restart* acceptance criterion; note that this comparison is particularly interesting, since ILS with restarts can be seen as an extreme case of HILS: it corresponds to an outer ILS loop that uses a perturbation size equal to the instance size (that is, it generates a new random solution) and the *RandomWalk* acceptance criterion.

Finally, note that in HILS(2) one additional parameter arises: the execution time given to the inner ILS algorithm. In fact, this type of parameter arises in any similar hybrid where a metaheuristic is used as an improvement method and we will set this parameter empirically.

# 4 Benchmark Instances

While most of the studies of algorithms for the QAP are based on instances from QAPLIB, in this study we use benchmark instances that we have randomly generated. There are a few reasons for this. A first is that in this way we can systematically vary instance characteristics. Secondly, in this way we can generate a larger set of reasonably uniform instances, which can be used, for example, for automated tuning tools and allow for the comparison on test instances. Thirdly, this allows also to study the algorithm behavior on larger instances, since most QAPLIB instances are of rather small size.

In this study, we generated a new set of QAP instances similar to some types of instances used in the research of the Metaheuristics Network [17]. These instances are generated such that they have widely different distributions of the entries of the flow matrices. In particular, the flow matrices are generated such that their entries are similar to the type of distributions found in real-world QAP instances [19].

The distance matrices are generated using two possibilities: Euclidean distances and grid distances.

**Euclidean distances.** Using this approach, the distance matrix entries are generated as the pairwise Euclidean distances between $n$ points in the plane:

1. each coordinate of each of $n$ points is generated randomly according to a uniform distribution in $[0, k]$.
2. The Euclidean distance between each pair of two points is calculated, rounded to the nearest integer, and returned as the distance matrix entry.

**Grid distances.** In this approach, the distance matrix corresponds to the Manhattan distance between the points on a $A \times B$ grid.

Instances generated based on Euclidean distances will have most probably only one single optimal solution, while instances generated based on Manhattan distances have at least four optimal solutions, due to the symmetries in the distance matrix.

Flow matrix entries follow an exponential distribution, where a large number of entries have small values, and very few are with large values. The flow matrices of all instances are symmetric and have a null diagonal.

**Structured flows.** Instances with structured flows are generated as follows:

1. $n$ points are generated randomly according to a uniform distribution in a square with dimension $100 \times 100$.
2. If the distance between two points $i$ and $j$ is above a threshold value $p$, the flow is zero.
3. Otherwise, the flow is calculated as $x = -1 \cdot \log(R)$, where $R$ is a random number in $[0, \cdots, 100]$, and $\mathit{flow} = \min\left\{100 \cdot x^{2.5}, 10000\right\}$

The numbers have been chosen in such a way that the objective values can still be represented as unsigned integers on a 32-Bit system.

As a result, we have two different levels of distance matrices (Euclidean (ES) and grid distances (GS)). For our experiments, we have generated for each level of distance matrices three different levels of flow matrices. Each of these three levels of flow matrices differs mainly in their sparsity (defined as the percentage of zero entries) and the resulting flow dominance values (which corresponds to the variation coefficient of the flow matrix entries multiplied by 100). Statistics on these values (including also the distance dominance) for each resulting instance class are given in Table 1. Of each of these classes, we have generated 100 instances for tuning and 100 for testing. The statistics are given for two instance sizes.

Table 1: Basic statistics on some instance classes of the benchmark instances. Given are the distance dominance (dd), the flow dominance (fd), and the sparsity of the flow matrix (sp).

| Class | size | dd | fd | sp | Class | size | dd | fd | sp |
|---|---|---|---|---|---|---|---|---|---|
| *ES*.1 | 100 | 47.17 | 274.57 | 0.20 | *GS*.1 | 100 | 69.31 | 275.37 | 0.21 |
| *ES*.2 | 100 | 47.17 | 386.28 | 0.45 | *GS*.2 | 100 | 69.31 | 385.57 | 0.43 |
| *ES*.3 | 100 | 47.15 | 493.25 | 0.90 | *GS*.3 | 100 | 69.31 | 492.93 | 0.89 |
| *ES*.1 | 200 | 47.41 | 276.27 | 0.21 | *GS*.1 | 200 | 68.98 | 276.42 | 0.20 |
| *ES*.2 | 200 | 47.41 | 386.59 | 0.45 | *GS*.2 | 200 | 68.98 | 386.26 | 0.42 |
| *ES*.3 | 200 | 47.42 | 496.29 | 0.91 | *GS*.3 | 200 | 68.98 | 494.12 | 0.92 |

# 5 Experimental Results

## 5.1 Experimental setup

All the experiments that we report here have been run on Intel Xeon 2.4 Ghz quad-core CPUs with 6 MB cache and 8 GB of RAM running under Cluster Rocks Linux. Due to the sequential implementation of the algorithms only a single core is used for computations. If nothing else is said, the stopping criteria for the experiments on the various instance classes were based on the CPU time that is taken by our reimplementation of the robust tabu search algorithm of Taillard [18] to perform $1000 \cdot n$ iterations. The algorithms are coded in `C` and compiled with `gcc` version 3.4.6.

## 5.2 Choice of local search

As a first step, we identified the most appropriate local search to be used in ILS and HILS(2). To do so, we used a set of 30 instances of size 100 and we performed tests considering the following four options:

1. first-improvement two-exchange local search without don't look bits.
2. first-improvement two-exchange local search with don't look bits.
3. first-improvement two-exchange local search with don't look bits and resetting of don't look bits during perturbation.
4. best-improvement two-exchange local search.

Since ILS is the main part of HILS(2), the performance of local search was studied using exclusively an ILS algorithm. A statistical analysis of the experimental results showed that the third option was the most performing one. In this option, after a perturbation only the don't look bits of the units that have changed the position are reset to zero.

## 5.3 Experimental study of HILS combinations

As a next step, we generated systematically a number of HILS(2) variants. In total, eight variants have been studied that were obtained by all possible combinations of the following settings. The outer and the inner ILS can use the acceptance criteria *Better* or *RandomWalk* and the length of the inner ILS is terminated after either $n$ or $2n$ ILS iterations, where $n$ is the instance size. The perturbation sizes were set, following the rationale given in Section 3.3, as follows. The outer ILS used variable perturbation sizes taken from the set $\{0.25n, 0.5n, 0.75n\}$, while the inner ILS varied the perturbation size among the integers in the interval $[3, 0.25n - 1]$; in both cases, the changes in the perturbation sizes follow the rules of the basic VNS [8].

Each of the resulting eight HILS(2) algorithms was applied to 100 instances of the above mentioned benchmark set of sizes 100 and 300. The results of the experiments were analyzed using pairwise Wilcoxon tests using Holm's correction to identify the winning combinations. While we expected to observe clear trends concerning the winning configurations, no fully conclusive results were obtained. In fact, seven of the eight configurations have been among the best performing configurations (taken into account statistically not significantly different performance). The only slight trend appeared to be that on less sparse instances, for the outer ILS the *Better* acceptance criterion appeared to be preferable, while for very sparse instances the *randomWalk* criterion gave overall better performance.

## 5.4 Performance comparison of HILS and ILS variants

Given that no fully conclusive results were obtained in our previous analysis, as a next step, we fine-tuned two ILS algorithms that were used as building blocks of HILS(2) (ILS with better acceptance criterion (ILS$b$), ILS with random walk acceptance criterion (ILS$rw$)), as well as the ILS algorithm with *Restart* acceptance criterion (ILS$rr$). For this task, we used F-race, a racing algorithm based on the Friedman two-way analysis of variance [1]; for each tuning task a same maximum number of 6 000 algorithm evaluations was given as the computational budget for F-race.

The tuning of ILS$b$, ILS$rw$, and ILS$rr$ was considered to allow for a fair comparison. For the first two variants, only one parameter was tuned; the perturbation scheme, while for ILS$rr$, an additional parameter, the number of iterations without improvement before a random restart is triggered ($i_r$), also needs to be tuned. Perturbation schemes considered are a fixed perturbation scheme and variable neighborhood search (VNS). For each scheme, several choices of perturbation strength ($k$), step size ($k+$), and maximum perturbation strength ($k_{max}$) are available. The winning configurations from F-Race are then used when comparing HILS(2) to the underlying ILS variants. Table 2 shows the parameter settings selected by F-Race for the three basic ILS variants.

Four parameters were considered for tuning HILS(2); the perturbation scheme, the acceptance criterion for the outer loop ($Acc_o$) and the inner loops ($Acc_i$), and the length of the inner loop ($L$) as a multiple of the instance size. The perturbation scheme consists of several subsidiary parameters; perturbation direction $P$, which controls how the perturbation size changes (increasing, decreasing, or fix), minimum perturbation strength for the outer loop, $k_{min-o}$, change in perturbation strength after every iteration $k+$, maximum perturbation strength for outer loop, $k_{max-o}$, and maximum perturbation strength for inner loop, $k_{max-i}$. Acceptance criterion $Acc_o$ and $Acc_i$ consist each of two choices as explained previously: *Better* ($b$) and *RandomWalk* ($rw$). The parameter settings of HILS(2) chosen by F-Race is summarized in Table 3.

On most of ES instances, the fixed perturbation scheme was performing best, while for GS instances, the VNS scheme seems to be the better choice. Intuitively, if a fixed perturbation scheme is chosen, the perturbation strength has to be very large to allow stronger diversification by the algorithm. Results from F-Race confirm this since the surviving perturbation strengths for a fixed perturbation size are always large: $0.7n$ or $0.9n$. For the VNS scheme, several $k_{min-o}$ values were considered during tuning, and all the values, $n/6$, $n/5$, and $n/4$ are chosen as good parameter settings for respective instance class. F-Race suggests a long inner loop length for ES instances, while for GS instances, a shorter inner loop length was chosen all the time.

The results of the comparison between HILS(2) and the three ILS algorithms are presented next. For each combination of instance class and size, we first computed the average solution quality across the 100 test instances and we used them as the basis of our comparison. The percentage difference between these values and results obtained by other algorithms are then computed. In Table 4 we give the results obtained by HILS(2) and the underlying ILS variants. Pairwise Wilcoxon tests with Holm corrections for multiple comparisons were conducted to check the statistical significance of our results. If the p-value is lower than our chosen level of $\alpha = 0.05$, the difference between the corresponding algorithms is deemed significant, and printed in italics.

The result of this experiment show very promising performance of HILS(2). HILS(2) has obtained the lowest average for 11 out of 12 instance classes; in many cases the observed differences are statistically significant. Only on dense ES instance set of size 100, the average obtained by

Table 2: Tuned parameter settings for basic ILS variants. The two entries for each parameter correspond to instances of size 100 and 200. See the text for more details.

| | $Pert.$ | | $k_{min}$ | | $k+$ | | $k_{max}$ | | $ir$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Instances*** | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 |
| | | | | | ILS$b$ | | | | | |
| $ES.1$ | fix | fix | 40 | 80 | 0 | 0 | 40 | 80 | - | - |
| $ES.2$ | fix | fix | 40 | 80 | 0 | 0 | 40 | 80 | - | - |
| $ES.3$ | VNS | VNS | 3 | 3 | 3 | 3 | 90 | 180 | - | - |
| $GS.1$ | VNS | VNS | 3 | 3 | 3 | 1 | 90 | 180 | - | - |
| $GS.2$ | VNS | fix | 3 | 20 | 1 | 0 | 90 | 20 | - | - |
| $GS.3$ | VNS | VNS | 3 | 3 | 3 | 1 | 90 | 180 | - | - |
| | | | | | ILS$rw$ | | | | | |
| $ES.1$ | VNS | VNS | 3 | 3 | 3 | 3 | 90 | 180 | - | - |
| $ES.2$ | VNS | VNS | 3 | 3 | 3 | 3 | 90 | 180 | - | - |
| $ES.3$ | VNS | fix | 3 | 140 | 1 | 0 | 30 | 140 | - | - |
| $GS.1$ | VNS | fix | 3 | 20 | 3 | 0 | 90 | 20 | - | - |
| $GS.2$ | VNS | fix | 3 | 20 | 3 | 0 | 90 | 20 | - | - |
| $GS.3$ | fix | fix | 40 | 140 | 0 | 0 | 40 | 140 | - | - |
| | | | | | ILS$rr$ | | | | | |
| $ES.1$ | fix | fix | 10 | 80 | 0 | 0 | 10 | 80 | 1 | 1 |
| $ES.2$ | fix | fix | 10 | 80 | 0 | 0 | 10 | 80 | 1 | 2 |
| $ES.3$ | VNS | VNS | 3 | 3 | 3 | 3 | 90 | 180 | 1 | 1 |
| $GS.1$ | VNS | fix | 3 | 80 | 3 | 0 | 90 | 80 | 1 | 1 |
| $GS.2$ | VNS | VNS | 3 | 3 | 3 | 1 | 90 | 90 | 1 | 1 |
| $GS.3$ | VNS | VNS | 3 | 3 | 1 | 1 | 90 | 90 | 10 | 2 |

ILS$_{rw}$ is very slightly less than that of HILS(2) (the truncation to two significant positions hides this fact in the table), but not in a statistically significant way.

## 5.5 Comparison of HILS, ILS-ES, ILSts, and RoTS

As a next step in our analysis, we compared the performance of HILS(2) to three well known algorithms from literature. The first is robust tabu search (RoTS) by Taillard [18], an algorithm, which is chosen as a baseline in many comparisons of SLS algorithms, iterated tabu search (ILSts), variants of which have been discussed in several articles [15, 5, 12, 13], and a state-of-the-art population-based ILS variant, the ILS-ES from [16]. All algorithms are re-implemented and run for a same computation time. The results of the comparison are given in Table 5. As in the previous comparison, we normalize the results w.r.t. the best performing algorithm and indicate

Table 3: Parameter settings for HILS(2). The two entries for each parameter correspond to instances of size 100 and 200. See the text for more details and an explanation of the table entries.

| | $P$ | | $k_{min-o}$ | | $k+$ | | $k_{max-o}$ | | $k_{max2}$ | | $Acc_o$ | | $Acc_i$ | | $L$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Instances*** | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 | 100 | 200 |
| $ES.1$ | VNS | fix | 25 | 140 | 25 | 0 | 75 | 140 | 25 | 140 | rw | b | rw | rw | 30 | 30 |
| $ES.2$ | fix | fix | 70 | 140 | 0 | 0 | 70 | 140 | 70 | 140 | rw | b | rw | rw | 30 | 10 |
| $ES.3$ | fix | fix | 90 | 180 | 0 | 0 | 90 | 180 | 90 | 180 | b | b | rw | rw | 10 | 2 |
| $GS.1$ | VNS | VNS | 17 | 34 | 17 | 34 | 83 | 167 | 17 | 34 | b | rw | b | rw | 2 | 2 |
| $GS.2$ | VNS | VNS | 20 | 50 | 20 | 50 | 80 | 150 | 20 | 50 | rw | rw | b | rw | 2 | 5 |
| $GS.3$ | VNS | fix | 17 | 180 | 17 | 0 | 83 | 180 | 17 | 180 | b | b | b | rw | 2 | 5 |

Table 4: Comparison between HILS, ILS$b$, ILS$rw$, and ILS$rr$. Given are the percentage deviations from the best average across the 100 test instances. Statistically significant differences are indicated in italics font. See the text for details.

| ***Class*** | ***Size*** | ***t*** | HILS | ILS$b$ | ILS$rw$ | ILS$rr$ |
|---|---|---|---|---|---|---|
| ES.1 | 100 | 54 | 0.00 | *0.33* | 0.00 | *0.22* |
| ES.2 | 100 | 54 | 0.00 | *0.43* | *0.06* | *0.21* |
| ES.3 | 100 | 54 | 0.00 | *1.07* | *0.94* | 0.07 |
| GS.1 | 100 | 54 | 0.00 | *0.08* | *0.02* | *0.01* |
| GS.2 | 100 | 54 | 0.00 | *0.11* | *0.02* | *0.02* |
| GS.3 | 100 | 54 | 0.00 | *0.03* | *0.26* | *0.02* |
| ES.1 | 200 | 438 | 0.00 | *0.26* | 0.02 | *0.23* |
| ES.2 | 200 | 438 | 0.00 | *0.27* | 0.00 | *0.24* |
| ES.3 | 200 | 438 | 0.00 | *0.38* | *0.49* | 0.07 |
| GS.1 | 200 | 438 | 0.00 | *0.26* | *0.08* | *0.14* |
| GS.2 | 200 | 438 | 0.00 | *0.45* | *0.05* | *0.02* |
| GS.3 | 200 | 438 | 0.00 | 0.02 | *0.33* | 0.68 |

significantly worse performance according to the Wilcoxon test with Holm's corrections for multiple comparisons in italics font. On instance size 100, ILS-ES is clearly the best performing algorithm: it obtains significantly better solutions on every instance class considered. For larger instances, HILS(2) shows a particularly good performance on dense instances. Only on very sparse instances (those with class identifier 3), ILS-ES shows (slightly) better average results. This behavior is true on instance sizes 200, 300, and 500.

### 5.6 HILS(3)

As a final experiment, we conducted tests on structured Taillard's instances (taiXXe instances) of size 125, 175, and 343 [6]. These instances were designed to be hard for local search algorithms by integrating some block structures in the flow and distance matrices. We selected one instance per size and conducted 20 independent trials for each algorithm. Since we found rather poor behavior of HILS(2) on this class of instances, on which it is known that large perturbations are needed, we tested also an ad-hoc variant of an HILS(3) algorithm. This HILS(3) was using at each of the three hierarchy levels a *Better* acceptance criterion. The perturbation size was at each level modified in a VNS fashion using as possible perturbation strengths in the outermost ILS the set $\{0.25n, 0.5n, 0.75n\}$, in the middle-level ILS the set $\{11, 12, .., 0.25n\}$, and in the innermost level the set $\{3, 4, .., 10\}$. The innermost ILS was terminated after ten iterations, while the middle-level ILS was terminated after five iterations. As can be seen in Table 6, HILS(3) resulted to be the best performing algorithm for these instances, even performing better than the ILS-ES approach. An illustration of the development of the solution quality over time is given in Figure 1.

## 6 Conclusions

In this article, we have introduced an example of a metaheuristic that has been successfully hybridized with itself, which shows that hybrid metaheuristics are potentially not only limited to hybrids among *different* metaheuristics. We illustrated this concept of self-hybridization using the example of iterated local search, where this concept can be applied in a reasonably straightforward way. In fact, the possibility of having nested local searches was already suggested in an earlier overview article on ILS [10], although there the idea of a self-hybridization of ILS has not been further elaborated. We further have shown that this idea of nesting iterated local searches leads to a hierarchy of ILS algorithms.

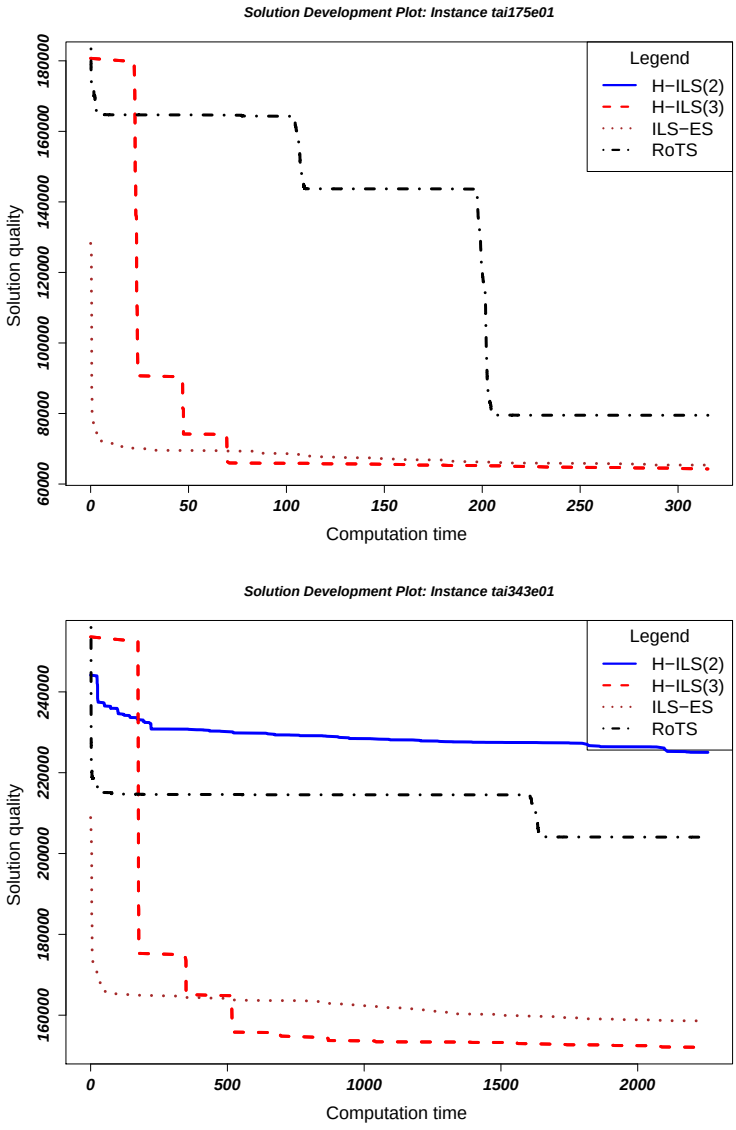


Figure 1: Plot of the development of the solution cost over computation time on two taiXXe instances. Given are the plots for HILS(2), HILS(3), ILS-ES and robust tabu search (RoTS).

Table 5: Comparison between HILS(2), ILS-ES, ILSts, and RoTS. The computation times for each instance class and all algorithms are indicated in the column ***t***. Given are the percentage deviations from the best average across the 100 test instances. Statistically significant differences are indicated in italics font.

| ***Class*** | ***Size*** | ***t*** | HILS(2) | ILS-ES | ILSts | RoTS |
|---|---|---|---|---|---|---|
| ES.1 | 100 | 54 | *0.08* | 0.00 | *0.24* | *0.26* |
| ES.2 | 100 | 54 | *0.06* | 0.00 | *0.26* | *0.26* |
| ES.3 | 100 | 54 | *0.27* | 0.00 | *0.30* | *0.95* |
| GS.1 | 100 | 54 | *0.03* | 0.00 | *1.99* | *0.29* |
| GS.2 | 100 | 54 | *0.02* | 0.00 | *1.79* | *0.28* |
| GS.3 | 100 | 54 | *0.01* | 0.00 | *5.16* | *0.42* |
| ES.1 | 200 | 438 | 0.00 | *0.11* | *0.11* | *0.20* |
| ES.2 | 200 | 438 | 0.00 | *0.09* | *0.14* | *0.17* |
| ES.3 | 200 | 438 | *0.27* | 0.00 | *1.31* | *0.94* |
| GS.1 | 200 | 438 | 0.00 | 0.01 | *0.65* | *0.34* |
| GS.2 | 200 | 438 | 0.00 | 0.03 | *0.66* | *0.30* |
| GS.3 | 200 | 438 | *0.07* | 0.00 | *1.17* | *0.72* |
| ES.1 | 300 | 1400 | 0.00 | *0.17* | *0.08* | *0.22* |
| ES.2 | 300 | 1400 | 0.00 | *0.16* | *0.17* | *0.24* |
| ES.3 | 300 | 1400 | *0.32* | 0.00 | *1.15* | *0.77* |
| GS.1 | 300 | 1400 | 0.00 | *0.13* | *1.04* | *0.39* |
| GS.2 | 300 | 1400 | 0.00 | *0.14* | *0.99* | *0.28* |
| GS.3 | 300 | 1400 | *0.06* | 0.00 | *2.52* | *1.53* |
| ES.1 | 500 | 7300 | 0.00 | *0.48* | *0.54* | *0.64* |
| ES.2 | 500 | 7300 | 0.00 | *0.53* | *0.36* | *0.67* |
| ES.3 | 500 | 7300 | *0.19* | 0.00 | *0.55* | *3.25* |
| GS.1 | 500 | 7300 | 0.00 | *1.70* | *6.95* | *5.27* |
| GS.2 | 500 | 7300 | 0.00 | *2.02* | *6.87* | *5.20* |
| GS.3 | 500 | 7300 | *0.02* | 0.00 | *2.91* | *2.18* |

In an experimental study, we have examined the performance of two nested ILS algorithms using the example application to the quadratic assignment problem. Computational results on large QAP instances have shown that after appropriate tuning, the HILS(2) algorithm tested reaches typically a significantly better performance than the underlying ILS algorithms from which it was derived. Experiments with an hierarchical ILS algorithm that is an example of the third level of the hierarchy, have shown promising results on a class of strongly structured instances that are designed to be hard for local search algorithms.

In summary, we hope that (i) this contributions inspires researchers to consider similar types of hybrid approaches and that (ii) it encourages them to test the hierarchical ILS approach, in particular, for a variety of other applications.

**Acknowledgments.**

This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Mohamed Saifullah Hussin acknowledges support from the Universiti Malaysia Terengganu. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate.

Table 6: Comparison between HILS(2), HILS(3), ILS-ES, and RoTS on three taiXXe instances. Given are the percentage deviations from the best average for each test instance.

| ***Instance*** | HILS(2) | HILS(3) | ILS-ES | RoTS |
|---|---|---|---|---|
| tai125e01 | 157.00 | 0.00 | 1.00 | 25.00 |
| tai175e01 | 220.00 | 0.00 | 2.00 | 23.00 |
| tai343e01 | 59.00 | 0.00 | 4.00 | 34.00 |

# References

[1] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, USA, 2002.

[2] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis. The quadratic assignment problem. In P. M. Pardalos and D.-Z. Du, editors, *Handbook of Combinatorial Optimization*, volume 2, pages 241–338. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[3] Call for Papers. HM2009: 6th International Workshop on Hybrid Metaheuristics. `http://www.diegm.uniud.it/hm2009/`, 2009.

[4] E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[5] J.-F. Cordeau, G. Laporte, and F. Pasin. Iterated tabu search for the car sequencing problem. *European Journal of Operational Research*, 191(3):945–956, 2008.

[6] Z. Drezner, P. Hahn, and É. D. Taillard. A study of quadratic assignment problem instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 174:65–94, 2005.

[7] I. Essafi, Y. Mati, and S. Dauzère-Pèréz. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35(8):2599–2616, 2008.

[8] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[9] H. H. Hoos and T. Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, USA, 2005.

[10] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[11] M. Lozano and C. García-Martínez. An evolutionary ILS-perturbation technique. In M. J. Blesa et al., editors, *HM 2008*, volume 5296 of *LNCS*, pages 1–15. Springer Verlag, Heidelberg, Germany, 2008.

[12] A. Misevicius. Using iterated tabu search for the traveling salesman problem. *Information Technology and Control*, 32(3):29–40, 2004.

[13] A. Misevicius, A. Lenkevicius, and D. Rubliauskas. Iterated tabu search: an improvement to standard tabu search. *Information Technology and Control*, 35(3):187–197, 2006.

[14] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[15] K. Smyth, H. H. Hoos, and T. Stützle. Iterated robust tabu search for MAX-SAT. In Y. Xiang and B. Chaib-draa, editors, *AI 2003*, volume 2671 of *LNAI*, pages 129–144. Springer Verlag, Heidelberg, Germany, 2003.

[16] T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(1):1519–1539, 2006.

[17] T. Stützle and S. Fernandes. New benchmark instances for the QAP and the experimental analysis of algorithms. In J. Gottlieb and G. Raidl, editors, *EvoCOP 2004*, volume 3004 of *LNCS*, pages 199–209. Springer Verlag, Heidelberg, Germany, 2004.

[18] É. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4–5):443–455, 1991.

[19] É. D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.