Computing and Informatics, Vol. 37, 2018, 656–672, doi: 10.4149/cai_2018_3_656

BREAKOUT LOCAL SEARCH FOR THE TRAVELLING SALESMAN PROBLEM

Mehdi El Krari

Faculty of Science, Mohammed V University in Rabat
Computer Science Laboratory
Rabat, Morocco
e-mail: mehdi@elkrari.com

Belaïd Ahiod

Faculty of Science, Mohammed V University in Rabat
LRIT, Associated Unit to CNRST (URAC 29)
Rabat, Morocco
e-mail: ahiod@fsr.ac.ma

Bouazza El Benani

Faculty of Science, Mohammed V University in Rabat
Computer Science Laboratory
Rabat, Morocco
e-mail: elbenani@hotmail.com

Abstract. The travelling salesman problem (TSP), a famous NP-hard combinatorial optimisation problem (COP), consists of finding a minimum length tour that visits n cities exactly once and comes back to the starting city. This paper presents a resolution of the TSP using the breakout local search metaheuristic algorithm (BLS), which is based on the iterated local search (ILS) framework and improves it by introducing some fundamental features of several well-established metaheuristics such as tabu search (TS) and variable neighbourhood search (VNS). BLS moves from a local optimum of a neighbourhood to another by applying perturbation jumps whose type and number are determined adaptively. It has already been applied to many COP and gives good results. This innovative hybridisation resolved well 41 instances from the commonly used benchmark library TSPLIB. The high quality of experimental results shows the competitiveness of the proposed algorithm compared to other algorithms based on local search.

Keywords: Travelling salesman problem, breakout local search, adaptive perturbation strategy, iterated local search

1 INTRODUCTION

The travelling salesman problem (TSP) [1, 3] is one of the most universally studied combinatorial optimisation problems (COP). It is for more than half a century the focus of many researchers from all around the world. Work on the TSP had an enormous impact on the emergence and evolution of many important areas of research (stochastic local search [9], integer programming [22], complexity theory [11] ...). Besides its importance in practice, the TSP has also become a standard testbed for new algorithmic ideas. The problem was introduced for the first time in 1859 by William Rowan Hamilton. In its classic form, the statement is as follows: "A travelling salesman must visit once and only once a finite number of cities and return to its point of origin. Find the order of visiting cities that minimises the total distance travelled by the salesman."

Application domains of TSP are numerous: logistic problems of transportation of goods, as well as people, and more generally all kinds of scheduling problems. Some issues in the industry are modelled as a travelling salesman problem as the optimisation of trajectories of machine tools: How to drill several points on an electronic card as quickly as possible? The manufacturing of VLSI chips [23] and X-ray crystallography [21] are just a few examples of several applications of the TSP. Its simplicity and adaptability have made this problem for decades a starting point for more work and research. This COP belongs to NP-complete problems [11].

We introduce the breakout local search metaheuristic algorithm (BLS) for solving the TSP. While iterated local search (ILS) [19] may suffer from lack of effectiveness in escaping attractions, BLS follows the basic scheme of this framework and improves it by combining the features of other robust and efficient methods, including variable neighbourhood search (VNS) [15] adapted on perturbations [8]. The main idea of BLS is to use a descent-based local search to find local optima, and use the most appropriate perturbations in order to move (without being blocked) from one neighbourhood to another in the search space. Perturbation strategy of BLS is based on both the history and state of search; it introduces a variable degree of diversification by determining perturbation dynamically jumps and performing adaptive selection from several types of dedicated movements.

BLS was developed by Benlic and Hao in 2012. Since then, it has been used for solving some COP, such as the minimum sum coloring problem [4], maximum clique problems [5], quadratic assignment problem [7] and max-cut problem [6] and has given very good results. This paper deals with a resolution of the TSP, whose performance will be evaluated by solving 41 benchmark instances of the TSPLIB [16].

The reminder of this paper is organised as follows: Section 2 introduces the TSP and local search approaches. Section 3 describes in detail the BLS metaheuristic.

Section 4 first reports the computational results and comparisons, which are based on the TSPLIB benchmark instances; then it provides justification of the choice for some of BLS parameter settings. Section 5 is devoted to a discussion around BLS highlights which makes it different from other ILS algorithms. Finally, Section 6 concludes the paper.

2 THE TRAVELLING SALESMAN PROBLEM

2.1 TSP Formulation and Landscape Analysis

Given n cities and a matrix $D = (d_{ij})_{n\times n}$ of distances between all pairs of these cities. The TSP aims to find a shortest closed tour (i.e. Hamiltonian cycle) in which each city is visited once and only once. Each tour can be represented by a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of integers from 1 to n where $j = \pi(i)$ denotes the city j to visit at step i, $i = 1, 2, \ldots, n$. Therefore, the goal of TSP is to search a permutation π (tour) that minimises the tour length given by the following equation:

$$\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \tag{1}$$

where d_{ij} is the distance between city i and city j. In this paper, we consider the symmetric TSP where the distances satisfy $d_{ij} = d_{ji}$ for $1 \leq i, j \leq n$.

The study of TSP landscape [2] (or for any COP) allows us to know which operator exploits better the search space of the problem. Changing neighbourhood must be made such that it gives a new search area without escaping too far from the previous neighbourhood, in which case would provide an independent neighbourhood from those before. Such a study requires knowledge of some notions such as the fitness function and correlation.

As mentioned earlier, the most common fitness function is that using the length of the tour. A good fitness is a fitness with a low value, so a shorter tour. The landscape of the problem depends on the fitness function of the different solutions, hence the concept of fitness landscape.

The second important concept to know is that of correlation [14, 18], introduced to provide a measure of the difficulty of a problem taking in account some operators. It shows how much a tour is linked to its neighbours.

Experiments done previously on the TSP [10, 2, 18] showed that the landscape is more correlated for 2-opt move than others. That is why it has been chosen in this paper, especially in the local search phase.

2.2 Local Search Approaches for the TSP

Local search methods start from an initial configuration and apply successive transformations to the current solution while a stopping criterion is not verified. There-

fore, the implementation requires the choice of an initial solution(s) and local transformation(s), also known as moves. These algorithms are frequently used for solving the TSP problem. They improve iteratively the current solution seeking better in its predefined neighbourhood. The algorithm stops when he reaches a maximal number of iterations or when there is no better solution in a given neighbourhood: a local minimum is attained. Historically, 2-opt [9] is one of the first algorithms to solve instances of the TSP. It is a local search algorithm whose neighbourhood is defined by removing two non-adjacent edges of the current solution. The two parts obtained from this solution are reconnected by two other edges to obtain a new solution. The iterated local search (ILS) proposed by T. Stützle [19] is a framework based on local searches; it is a stochastic method that produces a sequence of solutions generated by an introduced heuristic, leading to better results than if we use repeated random testing of this heuristic. One of the main steps of ILS is perturbations [8], it is to avoid to be trapped in a local optimum by switching to another more distant. Local search in such a case will run more easily and will be more efficient.

However, perturbations applied by ILS for some TSP instances may not escape attraction from some neighbourhoods. Based on the later point, we introduced a new approach based on diversification of perturbations and moves, named "Breakout Local Search".

3 BREAKOUT LOCAL SEARCH FOR THE TSP

3.1 An Overview of BLS

The overall approach of BLS is a move from a local optimum of a neighbourhood to another one by applying perturbation jumps type and number of which are determined adaptively. Algorithm 1 below is a pseudo-code of the BLS algorithm for solving the TSP. BLS starts from an initial solution π_0 (having a cost C_0) and performs local search (the steepest descent) to reach a new local optimum π (lines 13–18). Each iteration of the local search algorithm browses the whole neighbourhood and chooses the best improving solution to replace the current neighbourhood solution. If no improvement is made in the neighbourhood, the local optimality is reached. BLS tries firstly to escape the neighbourhood attraction of the current local optimum to move to a new neighbourhood attraction (line 38). BLS applies then a number of moves (or jumps) starting from L_0 and dedicated to the current local optimum π which becomes disturbed and serves as a starting point for the next descent of the local search procedure. When the local search procedure returns the same neighbour π, BLS disturbs it more strongly by selecting a stronger perturbation, and (when different perturbations are not able to move to a neighbourhood with a new best local optimum) by increasing the number of jumps L by 1 to apply for this perturbation (lines 30–32). After visiting some neighbourhoods without improving the best solution found so far (lines 23–24) T times, BLS performs a much stronger perturbation with L_{max} jumps to permanently direct search into a new and more distant region in the search space (lines 26–30).

Algorithm 1 Breakout Local Search for TSP

Require: Maximal descents to perform $Desc_{max}$, initial number of jumps L_0, maximal consecutive visited local optima without any improvement T, number of jumps in strong perturbation L_{max}.

Ensure: A solution π_{best}

$S \leftarrow 0$
$\pi \leftarrow initialSolution()$ /* generates a solution with greedy or random algorithm */
$C \leftarrow Cost(\pi)$
$\pi_{best} \leftarrow \pi$ /* π_{best} saves best solution found */
$c_{best} \leftarrow c$ /* c_{best} saves best objective value */
$\omega \leftarrow 0$ /* ω gives the number of consecutive non-improving local optima */
$L \leftarrow L_0$ /* L saves number of jumps to perform, set to its minimal value L_0 */
$L\omega \leftarrow 0$ /* $L\omega$ is an indicator to guess which move to use in next perturbation */
$c_p \leftarrow c$ /* c_p saves objective value of last descent */
$Desc \leftarrow 0$ /* $Desc$ saves current number of descents */
$Iter \leftarrow 0$ /* global iteration counter */
while $Desc < Desc_{max}$ **do**
  while $\exists 2optMove(x, y)$ such that $(c + delta2Opt(\pi, x, y) < c)$ **do**
    $\pi \leftarrow \pi \oplus 2optMove(x, y)$ /* perform the best improving move */
    $c \leftarrow c + delta2Opt(\pi, x, y)$ /* cost variation of π with (x,y) move */
    $update\_H(Iter, x, y)$ /* update iteration number when edges move was last performed */
    $Iter \leftarrow Iter + 1$
  end while
  if $c < c_{best}$ **then**
    update π_{best} and c_{best}
    $\omega \leftarrow 0$
    $Desc \leftarrow Desc \times \frac{1}{2}$ /* reduce current number of descents */
  else
    $\omega \leftarrow \omega + 1$
  end if
  if $\omega > T$ **then** /* performing strong perturbation */
    $\pi \leftarrow dbmPerturb(\pi, L_{max})$ /* Double Bridge Move perturbation with L_{max} moves */
    $\omega \leftarrow 0$
    $Desc \leftarrow Desc \times \frac{7}{8}$ /* reduce current number of descents */
  else if $c = c_p$ **then** /* search returned the previous local optimum */
    $L\omega \leftarrow L\omega + 1$ /* increment indicator for the type of perturbation */
    $L \leftarrow L_0 + \frac{L\omega}{3}$ /* increment moves if the 3 moves do not improve */
  else /* Search escaped from the previous local optimum, reinitialize indicator */
    $L\omega \leftarrow 0$
  end if

36: $c_p \leftarrow c$ /* update the objective value of the previous local optimum */
37: **if** Strong perturbation was not performed **then**
38: $\pi \leftarrow Adaptive\_Perturbation(\pi, L, L\omega, H, Iter, \omega)$ /* see algorithm 4 */
39: **end if**
40: **end while**
41: **return** π_{best}

3.2 Exploring Solution Space by Neighbourhood

BLS is a metaheuristic based on the ILS framework, the process of descent/perturbation is redone as we have not yet reached a number ($Desc_{max}$) of descents. BLS uses the steepest descent with a 2-opt neighbourhood in local search and is called different perturbations, each of them introduces a different neighbourhood as shown in Algorithm 2. One of these perturbations (depending on $L\omega$ value) is applied L times to a local optimum π with moves chosen from the set of candidates M.

Algorithm 2 Dynamic_Perturbation($\pi, L, L\omega, H, Iter, \omega, M$)

Require: Initial solution π which is a local optimum, number of jumps L, indicator of perturbation type $L\omega$, matrix of history moves H, global iteration counter $Iter$, number of consecutive non-improving local optima ω, set of candidate moves M.
Ensure: A perturbed solution π
1: **if** ($L\omega$ mod 3) = 0 **then** /* Call Perturbation_Operator with 2Opt move */
2: $\pi \leftarrow Perturbation\_Operator(\pi, L, H, Iter, \omega, M, 2Opt)$
3: **else if** ($L\omega$ mod 3)= 1 **then** /* Call Perturbation_Operator with insert move */
4: $\pi \leftarrow Perturbation\_Operator(\pi, L, H, Iter, \omega, M, insert)$
5: **else** /* Call Perturbation_Operator with swap move */
6: $\pi \leftarrow Perturbation\_Operator(\pi, L, H, Iter, \omega, M, swap)$
7: **end if**
8: **return** π

Perturbations play a key role in BLS since the steepest descent cannot escape the local optimum. BLS tries then to exit the current neighbourhood by introducing different parameterised perturbations, starting with

1. the nodes/cities to move, and
2. how many times to perturb and
3. which move type to make.

All these steps will be detailed in the next sections.

3.3 Principle of Adaptive Perturbation

3.3.1 Main Idea

A variety of perturbations in BLS resides in the number of jumps and the (three) types of perturbations implemented. The number of jumps increases once all the

perturbations attempted without any improvement.

- The first perturbation is performed with a 2-opt move, wich is characterised by an exchange of two non-adjacent edges as it is shown in Algorithm 3.
- Secondly we perform an insert move. It comes to move a node from one position to another. This move results in a change of three edges.
- Finally a swap move. It comes to move an edge from one position to another. This move results in a change of four edges.

We present below in Algorithm 3 a perturbation launched by the 2-opt move. The same algorithm is adapted to *insert* and *swap* moves, by applying the appropriate move and saving the affected edges.

Algorithm 3 Perturbation_Operator$(\pi, L, H, Iter, \omega, M, mvt)$

Require: Initial solution π which is a local optimum, number of jumps L, matrix of history moves H, global iteration counter $Iter$, number of consecutive non-improving local optima ω, set of candidate moves M, move type mvt.
Ensure: A perturbed solution π

1: **for** $i \leftarrow 1, L$ **do**
2: take a pair $(x, y) \in M$
3: **if** $mvt = 2Opt$ **then**
4: $\pi \leftarrow \pi \oplus 2OptMove(x, y)$
5: $c \leftarrow c + delta2Opt(\pi, x, y)$
6: **else if** $mvt = insert$ **then**
7: $\pi \leftarrow \pi \oplus insertMove(x, y)$
8: $c \leftarrow c + deltaInsert(\pi, x, y)$
9: **else**
10: $\pi \leftarrow \pi \oplus swapMove(x, y)$
11: $c \leftarrow c + deltaSwap(\pi, x, y)$
12: **end if**
13: $update\_H(Iter, x, y)$
14: $Iter \leftarrow Iter + 1$
15: **if** $c < c_{best}$ **then**
16: update π_{best} and c_{best}
17: $\omega \leftarrow 0$
18: $Desc \leftarrow Desc \times \frac{1}{2}$
19: **end if**
20: **end for**
21: **return** π

Rather than performing random jumps all the time, BLS switches between three types of perturbations: directed, recency-based and random. Each perturbation generates, as shown in Algorithm 4, a set M of pairs that will be used in perturbations.

3.3.2 The Three Types of Perturbation Moves

The directed perturbation aims to build a set of candidates with the lowest degradation during perturbation move. These candidates should not have been solicited in the last γ moves: they are saved in a tabu list [12, 13], with the corresponding length γ. However an edge can be part of the tabu list but still selected if it leads to a solution that improves the best solution found so far. Directed perturbation is built based on the tabu list, and the quality of the moves to be applied. Eligible candidates for this perturbation are defined by the following set A:

$$A = \{\text{2OptMove}(u, v) \mid \min\{\text{delta2Opt}(\pi, u, v)\}, \\ (H_{u,v} + \gamma) < \text{Iter} \vee (\text{delta2Opt}(\pi, u, v) + c) < c_{best}, u \neq v\}. \quad (2)$$

The recency-based perturbation builds a set of candidates by using only the matrix H of historical moves. The moves are those which have been least recently used. These moves are identified by the set B such as:

$$B = \{\text{2OptMove}(u, v) \mid \min\{H_{uv}\}, u \neq v\}. \quad (3)$$

Finally, **the random perturbation** simply makes moves that are picked uniformly at random. Those moves are identified as:

$$C = \{\text{2OptMove}(u, v), u \neq v\}. \quad (4)$$

The three above formula adapt to both insert and swap movements, by applying the appropriate movement and delta operations.

Depending on the state of search, BLS selects one of these three perturbations pseudo-randomly with a probability. This state is determined by the parameter ω that gives the number of consecutive non-improving local minima. The aim is to give priority to the directed perturbation at the beginning of the search (when ω is still small), and reduce the chances of running when the neighbourhood has important attraction, to use other perturbations and have stronger diversifications.

We force the probability P of applying the directed perturbation to get values no smaller than a threshold P_0:

$$P = \begin{cases} e^{-\omega/T}, & \text{if } (e^{-\omega/T} > P_0), \\ P_0, & \text{otherwise.} \end{cases} \quad (5)$$

Given the probability P of using the directed perturbation, the probability of applying both the recency-based and the random perturbations is determined respectively by $(1 - P) \times Q$ and $(1 - P) \times (1 - Q)$ where Q is a constant from $[0, 1]$. Algorithm 4 links the three probabilities to their respective perturbation.

Algorithm 4 Adaptive_Perturbation$(\pi, L, L\omega, H, Iter, \omega)$

Require: A tour π which is a local optimum, number of jumps L, determinant of perturbation type $L\omega$, matrix of history moves H, global iteration counter $Iter$, number of consecutive non-improving local optima ω.

Ensure: A perturbed solution π

1: Determine probability P according to Formula (5) /* section above */
2: with a probability P, /* directed perturbation */
3: $\pi \leftarrow Dynamic\_Perturbation(\pi, L, L\omega, H, Iter, \omega, A)$
4: with a probability $(1 - P) \times Q$, /* recency-based perturbation */
5: $\pi \leftarrow Dynamic\_Perturbation(\pi, L, L\omega, H, Iter, \omega, B)$
6: with a probability $(1 - P) \times (1 - Q)$, /* random perturbation */
7: $\pi \leftarrow Dynamic\_Perturbation(\pi, L, L\omega, H, Iter, \omega, C)$
8: **return** π

3.3.3 Variation Jumps and Perturbation Moves

BLS varies (between two consecutive blocks) the number of jumps and performed moves. These changes are executed in a particular order: the first change performed is *2-opt* move, characterised by the exchange of two edges, the second change tried is *insert* move with three edges exchanged and finally the *swap* move defined by an exchange of four edges. The main idea is to move to the nearest neighbourhood and allow, at the same time, to escape the local optimum attractor and find a new and better solution. In the case where the three changes fail to escape from the optimum attractor, the number of jumps L is incremented.

The worst case is to redo the perturbation T times (consecutively) without being able to leave this attractor. A strong perturbation defined by L_{max} jumps with a Double Bridge [20] move is then performed. The current number of descents ($Desc$) is then reduced by $\frac{1}{8}$ in order to give a chance to the new neighbourhood to be enough scanned.

4 EXPERIMENTAL RESULTS

4.1 Experimental Protocol

BLS algorithm is programmed in Java 1.7, and compiled on a Pentium Dual-Core CPU T4400 with 2.20 GHz and 2.8 GB. 41 instances from the commonly used TSPLIB benchmark are considered in the experiments, their sizes range from 14 to 442 cities. Each instance is run 20 times, with the parameters listed in Table 1. The choice of parameter values was carried out after many preliminary tests. This is justified in Subsection 4.3.

We observed (in the worst cases) that BLS may never reach stopping conditions if maximum number of non-improving attractors visited T before strong perturbation is set to a small value: this deadlock is due to the repetitive reduction of current number of runs ($Desc$) which prevents reaching $Desc_{max}$. T must be greater than

Parameter	Value	Meaning
$Desc_{max}$	$50n$, $25n$	Maximal number of descents. $50n$ if $n < 200$, $25n$ otherwise
L_0	1	Initial jump magnitude
L_{max}	$0.5n$	Jump magnitude during strong perturbation
γ	n	Tabu tenure/length
P_0	0.75	Smallest probability to perform directed perturbation
Q	0.7	Probability to perform random over recency-based perturbation
T	$((Desc_{max} - 1)/8) + 1$	Maximal number of consecutive non-improving local minima

Table 1. Settings of important parameters

$(Desc - 1) \times (1 - \frac{7}{8})$, so we set T to $\frac{Desc_{max}-1}{8} + 1$.

4.2 Computational Results and Comparisons

In the following, there will be listed the used performance measures of BLS algorithm:

1. the average deviation of obtained solutions from the best known solution, denoted δ:
$$\delta = 100 \times (\bar{c} - bks)/bks\,[\%] \tag{6}$$
where $\bar{c}$ is the average tour length over 20 runs of BLS, and bks is the best known value which can be found in the TSPLIB [16];
2. the number of solutions of which the deviation does not exceed 1 % (over 20 runs), denoted $C_{1\,\%}$;
3. the number of solutions where the cost is equal to the best known solution – C_{opt}. Instances with a zero in the two (merged) columns means that all executions found the best known solution;
4. the CPU time in seconds.

We compare the performance of BLS with the standard local search using 2-opt neighbourhood [17] (LS 2-opt) from the literature, and basic ILS[1] with a 2-opt descent and a Double Bridge Move perturbation. The comparison reported in Table 2 is based on the Local Search framework used in BLS.

The results above shows the great contribution of BLS on both standard local search and ILS. 38 of the 41 instances did not exceed a deviation of 1 % at least once over the 20 executions, of which 30 have never exceeded. 27 instances have reached

[1] Thomas Stützle. TSP-TEST, Version 0.9. Available from `http://www.sls-book.net`, 2004.

Instance	n	bks	LS-2OPT (δ)	ILS		BLS		
				δ	$C_{1\,\%}/C_{opt}$	δ	$C_{1\,\%}/C_{opt}$	CPU Time (sec)
burma14	14	3 323	–		**0**		**0**	0.00
ulysses16	16	6 859	–		**0**		**0**	0.00
ulysses22	16	7 013	–		**0**		**0**	0.00
eil51	51	426	–	0.469	14/5	0.199	20/8	0.58
berlin52	52	7 542	–	0.106	19/19		**0**	0.00
st70	70	675	–	0.741	13/3		**0**	0.97
eil76	76	538	–	0.929	11/3	0.492	18/5	2.67
pr76	76	108 159	–	0.518	20/7		**0**	1.41
gr96	96	55 209	0.997	0.466	18/4	0.215	20/5	6.00
rat99	99	1 211	0.614	1.652	7/0	0.057	20/12	5.72
kroA100	100	21 282	0.073	0.282	20/8		**0**	4.02
kroB100	100	22 141	0.379	0.632	14/5	0.012	20/18	6.58
kroC100	100	20 749	0.546	0.882	13/2		**0**	4.38
kroD100	100	21 294	1.538	0.977	13/0	0.053	20/14	5.16
kroE100	100	22 068	0.983	0.770	15/1	0.164	20/5	5.89
rd100	100	7 910	0.961	0.619	14/3	0.010	20/16	10
eil101	101	629	1.657	1.749	3/0	1.017	11/0	6.07
lin105	105	14 379	0.642	0.285	19/7		**0**	2.10
pr107	107	44 303	0.093	0.950	9/0	0.042	20/11	5.89
pr124	124	59 030	0.953	0.281	19/7		**0**	7.91
bier127	127	118 282	0.649	0.686	15/1	0.139	20/4	31
ch130	130	6 110	0.999	1.244	7/0	0.324	20/2	14
pr136	136	96 772	–	1.775	7/0	0.675	20/0	15
gr137	137	69 853	0.824	1.266	10/0	0.347	20/0	20
pr144	144	58 537	–	0.091	20/7		**0**	4.39
ch150	150	6 528	–	1.241	8/0	0.412	20/2	18
kroA150	150	26 524	–	1.421	6/0	0.374	20/0	26
kroB150	150	26 130	–	1.309	8/0	0.237	20/1	22
pr152	152	73 682	–	0.415	19/1	0.030	20/8	15
u159	159	42 080	–	1.694	5/1		**0**	25
rat195	195	2 323	–	2.927	0/0	1.157	1/0	40
d198	198	15 780	–	0.754	16/0	0.581	20/0	46
gr202	202	40 160	–	1.805	0/0	1.314	5/0	35
ts225	225	126 643	–	0.706	16/15	0.110	20/10	50
gr229	229	134 602	0.911	1.306	6/0	1.403	2/0	49
gil262	262	2 378	1.099	2.397	0/0	1.345	6/0	70
a280	280	2 579	–	3.373	1/0	0.866	12/0	68
lin318	318	42 029	1.202	2.253	0/0	1.384	2/0	152
rd400	400	15 281	1.543	3.030	0/0	2.493	0/0	280
gr431	431	171 414	3.045	2.357	0/0	2.243	0/0	294
pcb442	442	50 778	5.185	3.096	0/0	2.315	0/0	205

Table 2. Comparative results between BLS and standard local search using 2-opt

at least once the optimum, of which 12 have always reached within a reasonable time.

4.3 Justification for Parameter Settings

The good quality of the results obtained by BLS is due in part to the choice of the parameters (see Table 1), each of these is justified by its role and influence in BLS. The most influential of these will be confirmed by comparative tests on three TSPLIB instances ($eil51$, $eil76$ and $eil101$). Results will be represented on two superposed charts: the first is a stacked bar charts, each stack gives the best, average and the worst solution. It will show only two results if the best found

solution is equal to the optimum. Second chart is a line chart showing evolution of running time of BLS. The first two parameters below are defined according to the instance size (N).

4.3.1 Maximum Number of Descents ($Desc_{max}$)

The maximum number of descents decides how many times the local search will be performed, the higher is the number of descents, the better are the results; each one is a new chance to find better tour or escaping the attraction. In return, the running time will be greater as shown in Figure 1. It was noted while testing that over a certain number of descents, the results become quite satisfactory and adding more only rises execution time without significant improvements.

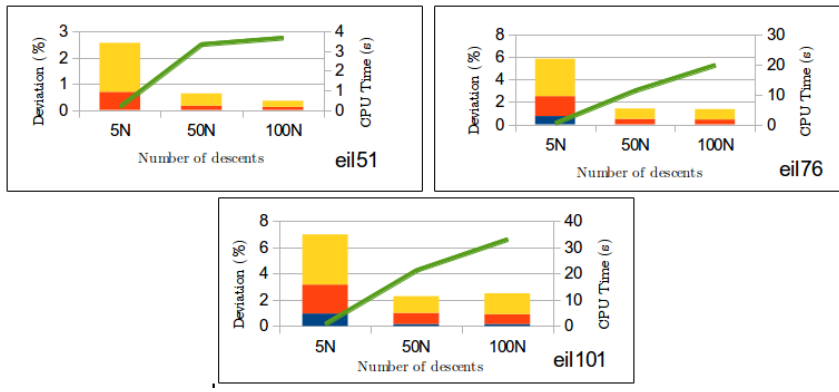

Figure 1. Varying number of descents for eil instances

4.3.2 Initial Number of Moves/Jumps (L_0)

This number gives the minimum of moves to be performed for each of the three types of perturbation; it has its impact on the change of neighbourhood and therefore the chances of unlocks. The larger it is, the more important is the changing neighbourhood, which may sometimes give the next descent a feeling of independence of the previous descents.

The best results were obtained as shown in Figure 2 when starting with a single jump, given the high effectiveness of diversification perturbations.

4.3.3 Maximum Number of Non-Improving Local Optima (T)

This variable indicates when we should perform the strong perturbation; it is reset to zero once this perturbation performed. The smaller T is, the higher is the number of perturbations, which increases the chances of unlocking. As mentioned earlier, the number T must be greater than $\frac{Desc_{max}-1}{8} + 1$ to avoid that the current number

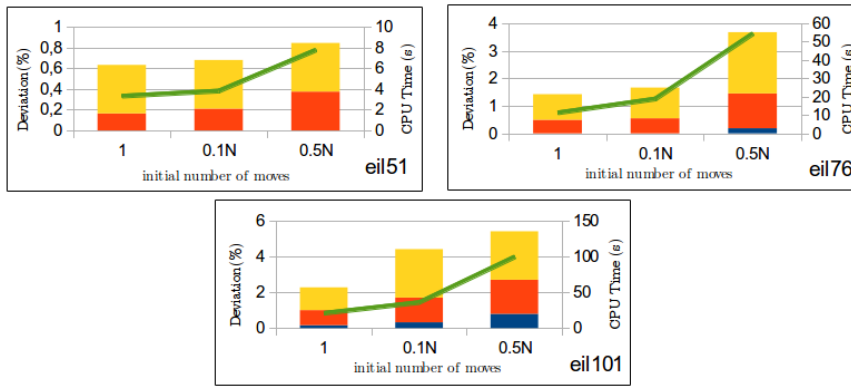

Figure 2. Varying initial number of moves for eil instances

of descents never reaches $Desc_{max}$. The tests above in Figure 3 are performed with variation of T three times: choosing firstly time the minimal value for each instance, and then doubling and tripling these values.

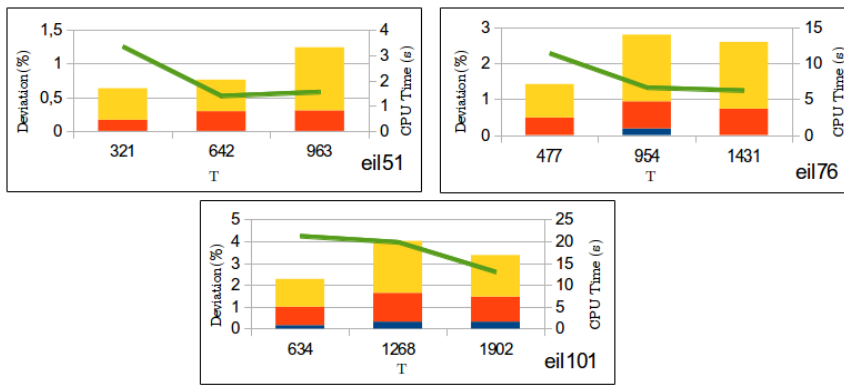

Figure 3. Varying maximum number of non-improving local optima for eil instances

4.3.4 Performing 3-Opt as Descent

BLS can be improved by changing neighbourhood in the steepest descent by switching to the 3-opt, which is larger due to the number of edges that are affected in each movement. This change will leave an impact on the performance of BLS: the history of changes is wider than the history constructed with the 2-opt steepest descent (three edges affected instead of two with 2-opt), then the perturbations become more diversified.

Table 3 shows BLS execution results with 3-opt (BLS 3-OPT), by comparing it with the standard local search using 3-opt neighbourhood (LS 3-OPT) [17] and BLS using 2-opt in the descent (BLS 2-OPT).

Instance	n	bks	LS-3OPT (δ)	BLS 2-OPT			BLS 3-OPT		
				δ	$C_{1\,\%}/C_{opt}$	CPU (*sec*)	δ	$C_{1\,\%}/C_{opt}$	CPU (*sec*)
burma14	14	3 323	–		**0**	0.00		**0**	0.00
ulysses16	16	6 859	–		**0**	0.00		**0**	0.00
ulysses22	16	7 013	–		**0**	0.00		**0**	0.00
eil51	51	426	–	0.199	20/8	0.58		**0**	2.69
berlin52	52	7 542	–		**0**	0.00		**0**	0.47
st70	70	675	–		**0**	0.97		**0**	11
eil76	76	538	–	0.492	18/5	2.67		**0**	8.55
pr76	76	108 159	–		**0**	1.41		**0**	21
gr96	96	55 209	0.997	0.215	20/5	6.00		**0**	23
rat99	99	1 211	0.614	0.057	20/12	5.72	0.033	20/12	16
kroA100	100	21 282	0.073		**0**	4.02		**0**	18
kroB100	100	22 141	0.379	0.012	20/18	6.58		**0**	62
kroC100	100	20 749	0.546		**0**	4.38		**0**	25
kroD100	100	21 294	1.538	0.053	20/14	5.16		**0**	35
kroE100	100	22 068	0.983	0.164	20/5	5.89	0138	20/10	52
rd100	100	7 910	0.961	0.010	20/16	10		**0**	54
eil101	101	629	1.657	1.017	11/0	6.07	0.063	20/12	39
lin105	105	14 379	0.642		**0**	2.10		**0**	29
pr107	107	44 303	0.093	0.042	20/11	5.89		**0**	36
pr124	124	59 030	0.953		**0**	7.91		**0**	21
bier127	127	118 282	0.649	0.139	20/4	31	0.102	20/9	247
ch130	130	6 110	0.999	0.324	20/2	14	0.216	20/7	196
pr136	136	96 772	–	0.675	20/0	15	0.374	20/0	244
pr144	144	58 537	–		**0**	4.39		**0**	12
ch150	150	6 528	–	0.412	20/2	18	0.117	20/14	284
kroA150	150	26 524	–	0.374	20/0	26	0.162	20/0	400
kroB150	150	26 130	–	0.237	20/1	22	0.185	18/0	626
pr152	152	73 682	–	0.030	20/8	15	0.041	20/6	374
u159	159	42 080	–		**0**	25		**0**	287
rat195	195	2 323	–	1.157	1/0	40	0.499	20/0	624

Table 3. Comparative results between BLS and standard local search using 3-opt

3-Opt neighbourhood brings many improvements to BLS. All running instances did not exceed a deviation of 1 %, the worst average does not even exceed 0.5 %. Only 4 out of 30 instances could not reach the optimum, while 21 have always reached the optimum.

5 DISCUSSIONS

Observing the overall framework of ILS, BLS uses local search to get local optima, and perturbation to vary the search. However, BLS differentiates itself from most ILS algorithms by the combination of various perturbation strategies of different strengths, triggered according to the search status. As explained in Section 3.3, BLS uses a perturbation of weaker diversification with a higher probability P as the search progresses toward improved new local optima.

By neglecting the maximum number of descents set in our algorithm, BLS always succeeds in finding the optimal solution. Indeed, the BLS search space expands after each series of perturbations until it finds the neighbourhood that his local optimum is the optimal solution. Below in Table 4 are shown the best and average running

CPU time execution (in seconds) of some instances of a BLS execution where we ignored this stopping condition criteria.

Instances	n	*bks*	Best	Average	Instances	n	*bks*	Best	Average
eil51	51	426	0	2.2	**kroC100**	100	20 749	1.4	20
berlin52	52	7 542	0	0.3	**kroD100**	100	21 294	24	433
st70	70	675	0.6	13	**kroE100**	100	22 068	1.19	47
eil76	76	538	1.2	9.4	**rd100**	100	7 910	1.5	55
pr76	76	108 159	0.3	6.9	**eil101**	101	629	2	257
rat99	99	1 211	2.1	960	**lin105**	105	14 379	0.6	41
kroA100	100	21 282	0.5	35	**pr107**	107	44 303	0.6	78
kroB100	100	22 141	8.1	95	**pr124**	124	59 030	1.3	27

Table 4. Execution time required for BLS to find the optimum

6 CONCLUSION

We explained in this paper the breakout local search approach for solving the TSP. This algorithm uses the ILS framework and brings improvements in the perturbation: it performs a local search and a perturbation-based diversification phase (to jump from a local optimum to another one). The local search procedure uses the steepest descent with 2-opt move strategy. To visit a local optima of high quality, the jumps toward new neighbourhood are adaptively controlled according to the state of search. Perturbation is achieved by varying the type of moves and then the size of a jump and selecting the most fitting perturbation for each diversification period.

The quality of BLS results reported in Section 4, proves its competitiveness compared to other algorithms. The repeated constructions (on each jump) of the set of candidate couples (formula (2), (3) and (4)) lead to a slowness of BLS compared to its competitors. For a better compromise of results' quality and execution time, we kept the 2-opt algorithm in the local search step so that BLS will not be penalised by the 3-opt slowness as shown in the comparison of Table 3.

In order to overcome this problem of slowness, BLS can be improved by introducing accelerated descent from the same neighbourhood and implementing efficient data structures to reduce the searching time in the construction of all candidates.

REFERENCES

[1] Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, Berlin, Heidelberg, 1994.

[2] Stadler, P. F.—Schnabl, W.: The Landscape of the Traveling Salesman Problem. Physics Letters A, Vol. 161, 1992, No. 4, pp. 337–344, doi: 10.1016/0375-9601(92)90557-3.

[3] Cook, W. J.: In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, 2014.

[4] Benlic, U.—Hao, J.-K.: A Study of Breakout Local Search for the Minimum Sum Coloring Problem. In: Bui, L. T., Ong, Y. S., Hoai, N. X., Ishibuchi, H., Suganthan, P. N. (Eds.): Simulated Evolution and Learning (SEAL 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7673, 2012, pp. 128–137.

[5] Benlic, U.—Hao, J.-K.: Breakout Local Search for Maximum Clique Problems. Computers and Operations Research, Vol. 40, 2013, No. 1, pp. 192–206, doi: 10.1016/j.cor.2012.06.002.

[6] Benlic, U.—Hao, J.-K.: Breakout Local Search for the Max-Cut Problem. Engineering Applications of Artificial Intelligence, Vol. 26, 2013, No. 3, pp. 1162–1173.

[7] Benlic, U.—Hao, J.-K.: Breakout Local Search for the Quadratic Assignment Problem. Applied Mathematics and Computation, Vol. 219, 2013, No. 9, pp. 4800–4815, doi: 10.1016/j.amc.2012.10.106.

[8] Codenotti, B.—Manzini, G.—Margara, L.—Resta, G.: Perturbation: An Efficient Technique for the Solution of Very Large Instances of the Euclidean TSP. INFORMS Journal on Computing, Vol. 8, 1996, No. 2, pp. 125–133, doi: 10.1287/ijoc.8.2.125.

[9] Croes, G. A.: A Method for Solving Traveling-Salesman Problems. Operations Research, Vol. 6, 1958, No. 6, pp. 791–812, doi: 10.1287/opre.6.6.791.

[10] Fonlupt, C.—Robilliard, D.—Preux, P.: Fitness Landscape and the Behavior of Heuristics. Evolution Artificielle, Vol. 97, 1997.

[11] Garey, M. R.—Johnson, D. S.—Stockmeyer, L.: Some Simplified NP-Complete Problems. Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC '74), ACM, New York, NY, USA, 1974, pp. 47–63, doi: 10.1145/800119.803884.

[12] Glover, F.: Tabu Search – Part I. ORSA Journal on Computing, Vol. 1, 1989, No. 3, pp. 190–206, doi: 10.1287/ijoc.1.3.190.

[13] Glover, F.: Tabu Search – Part II. ORSA Journal on Computing, Vol. 2, 1990, No. 1, pp. 4–32, doi: 10.1287/ijoc.2.1.4.

[14] Hordijk, W.: A Measure of Landscapes. Evolutionary Computation, Vol. 4, 1996, No. 4, pp. 335–360, doi: 10.1162/evco.1996.4.4.335.

[15] Mladenović, N.—Hansen, P.: Variable Neighborhood Search. Computers and Operations Research, Vol. 24, 1997, No. 11, pp. 1097–1100, doi: 10.1016/S0305-0548(97)00031-2.

[16] Reinelt, G.: TSPLIB – A Traveling Salesman Problem Library. ORSA Journal on Computing, Vol. 3, 1991, No. 4, pp. 376–384, doi: 10.1287/ijoc.3.4.376.

[17] Blažinskas, A.—Lenkevičius, A.—Misevičius, A.: Modified Local Search Heuristics for the Symmetric Traveling Salesman Problem. Information Technology and Control, Vol. 42, 2013, No. 3, pp. 217–230, doi: 10.5755/j01.itc.42.3.1301.

[18] Arbel Krakhofer, B.: Local Optima in Landscapes of Combinatorial Optimization Problems. Master's thesis, University of Vienna, Austria, 1995.

[19] Lourenço, H. R.—Martin, O. C.—Stutzle, T.: Iterated Local Search. In: Glower, F. W., Kochenberger, G. A. (Eds.): Handbook of Metaheuristics. Springer,

International Series in Operations Research and Management Science, Vol. 57, 2003, pp. 320–353.

[20] Martin, O.—Otto, S. W.—Felten, E. W.: Large-Step Markov Chains for the Traveling Salesman Problem. Complex Systems, Vol. 5, 1991, No. 3, pp. 299–326.

[21] Bland, R. G.—Shallcross, D. F.: Large Travelling Salesman Problems Arising from Experiments in X-Ray Crystallography: A Preliminary Report on Computation. Operations Research Letters, Vol. 8, 1989, No. 3, pp. 125–128, doi: 10.1016/0167-6377(89)90037-0.

[22] Dantzig, G. B.: Discrete-Variable Extremum Problems. Operations Research, Vol. 5, 1957, No. 2, pp. 266–288, doi: 10.1287/opre.5.2.266.

[23] Lawler, E. L.—Lenstra, J. K.—Rinnooy Kan, A. H. G.—Shmoys, D. B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley and Sons, New York, 1985.

Mehdi El Krari is Ph.D. candidate at Mohammed V University in Rabat, Morocco. He is working on metaheuristics for combinatorial optimization problems related to logistics and transportation. His main interests are combinatorial optimisation, discrete algorithms, stochastic local search, evolutionary computation, etc.

Belaïd Ahiod is Professor in the Computer Science Department at Faculty of Science of the Mohammed V University in Rabat, Morocco. His research interests include NP-hard combinatorial optimization problems, multi-objective optimization, metaheuristics, nature-inspired algorithms, etc.

Bouazza El Benani is Professor in Mohammed V University of Rabat, Morocco, Department of Computing Science since 1994. He holds his Ph.D. in computer science from Montreal University, Canada. His research interests include artificial intelligence, software engineering, evolutionary algorithms, big data, metaheuristic algorithms, healthcare.