# The effects of Local Search on Random Key encoding schemes for the Traveling Salesperson Problem

Calin Georgescu
Delft University of Technology
Delft, The Netherlands
C.A.Georgescu@student.tudelft.nl

Chris Bras
Delft University of Technology
Delft, The Netherlands
C.S.Bras@student.tudelft.nl

Zhi-Yi Lin
Delft University of Technology
Delft, The Netherlands
Z.Lin-6@student.tudelft.nl

## ABSTRACT

Differential evolution (DE) is a popular and powerful optimization tool used by practitioners for various problems, both continuous and discrete. Using an originally real-valued algorithm like DE to discrete, combinatorial optimization problems requires an encoding from the solution space to the continuous space and a mapping from the real-valued encoding back to the solution domain. Random Keys (RK) encodings provide a powerful yet efficient method of performing these tasks while preserving the permutation property required by many problems. In this research, we consider three general-purpose RK encodings and enhance their performance by using Local Search (LS) heuristics. As a benchmark, we use the Traveling Salesperson Problem (TSP) and study two LS variation operators: the well-known `2-opt` and a novel node-exchange-based operator `2-ne` that alters the solution more strongly than its counterpart. We investigate the effects of LS on the DE algorithm in terms of optimal parameter settings, solution quality, and impact of the LS search budget, to build a comprehensive picture of the interaction between the search heuristics and the RK encodings. The results indicate that both heuristics behave well in similar areas of parameter space and that `2-opt` tends to perform better than `2-ne` because it introduces less perturbance. We also show that both approaches scale linearly with the budget used for LS in terms of runtime and that `2-ne` is the more expensive heuristic, which may also benefit less from increases in search budget.

## 1 INTRODUCTION

DE is one of the best studied evolutionary algorithm paradigms in the domain of continuous optimization tasks, with applications in operations research, engineering tasks, physics, and biology [4, 13]. DE has many appealing features, including the relative ease of implementation and significant potential for parallelization, that led to it becoming one of the most widely adopted solutions for difficult real-value multi-dimensional optimization problems such as Fuzzy Job Scheduling [7] and Economic Load Dispatch [11].

The effectiveness of DE has led to extensions in the domain of discrete applications. Combinatorial optimization (CO) problems that can be solved with the help of DE include shop scheduling [12, 18], graph colouring [6, 9], linear ordering problem [1, 2] and TSP [5, 8, 16]. This work focuses on one of the most popular problems in theoretical and algorithmic computer science: TSP. The choice of this benchmark is motivated by its applicability in real-world optimization settings and its structural properties that enable for efficient and explainable local search heuristics to be applied.

For DE to be effectively applied to structured CO problems, some mapping must be defined between the real-valued search space and the discrete problem variables. One such mapping is the RK encoding. In this work, we benchmark three RK encoding strategies: the standard RK was first introduced in 1994 [3], as well as two RK encodings proposed by Kromer et al. [8] that seek to reduce the volume and dimensionality of the search space, respectively. This choice is motivated by the improvements the authors found in comparing the novel encodings with the original for TSP. In addition, by comparing different encodings that differently approach the continous search space, we aim to obtain more insightful and generalizable results.

In this work, we propose an extension of the RK-based DE algorithm that focuses on introducing effective local search heuristics to improve the performance of the baselines. To do this, we use the standard 2-edge exchange heuristic, also known as 2-opt, and introduce a novel local search heuristic, 2-node exchange (2-ne), that shares many of the advantageous properties of 2-opt while introducing more perturbance in the search space.

These choices are motivated by both empirical and analytical factors. Empirically, the DE algorithm using the standard RK encoding has been observed to converge slowly when the variation of solutions is small, and converge erratically when variation increases. This suggests that a small variation coupled with iterative improvements of the intermediate solution might prove extremely effective. Theoretically, LS heuristics have been shown to produce state-of-the-art results for TSP in many Genetic Algorithm (GA) settings [10, 17]. However, the behavior of LS heuristics in the RK-based encoding space has not yet been studied. The goal of this research is to identify the effects of selected LS heuristics on the three RK-based continuous search spaces in a DE framework. To achieve this goal, we aim to answer the following research questions:

- **RQ**1. *Which combinations of parameters provide the most efficient and robust performance for the novel RKLS-based algorithms?*
- **RQ**2. *How does the introduction of LS into RK encodings influence the performance of the DE algorithm?*
- **RQ**3. *What effect does the budget of the LS subroutine have on the performance of the RK-based DE algorithm?*

The remainder of the paper is structured as follows: section 2 provides the essential background for the implementation of our algorithm. section 3 introduces and explains our proposed methods. The experimental setup and the results of the empirical study on selected instances are shown and analyzed in section 4.

## 2 BACKGROUND

This section provides the necessary background for the key concepts involved in the design, decision making, and implementation processes carried out in this work.

## 2.1 Traveling Salesperson Problem

TSP is one of the most famous problems in theoretical computer science, because of its contrastingly simple formulation and difficult computational nature. The problem considers a salesperson who wants to travel from their home and visit each city in a set of size $N$ before returning to the starting point, in the shortest route possible.

More formally, TSP is modelled as a weighted graph $G = (V, E, w)$ where the set of vertices $V$ contains all the cities the salesperson needs to visit and the edge set $E$ contains all the connections between the cities, together with their distance (also referred to as cost or weight), given by the function $w : E \to \mathbb{R}$. A solution to TSP is a permutation $\pi$ of the set $V$ and its cost is given by $c(\pi) = \sum_{i=0}^{n-2} w(v_i, v_{i+1}) + w(v_{n-1}, v_0)$. TSP is a minimization problem, where the goal is to find the optimal permutation: $\text{argmin}_\pi\, c(\pi)$. The computational complexity of TSP stems from the number of feasible solutions present in search space. In a symmetric setting, an enumeration-based algorithm would require $\mathbb{O}((|V| - 1)!)$ operations to generate and evaluate all possible solutions.

More nuanced versions of TSP exist, such as variations that introduce asymmetric weights between cities, but in this research we focus on the most commonly occurring version, using datasets from TSPLIB [14].

## 2.2 Random Key Encodings

Random key encodings are efficient and straightforward mappings for permutation representations, used primarily in real-valued metaheuristic algorithms. First proposed by Bean et al. [3] in 1994, RK encodings have since been extended and specialized for specific use cases, with use cases in both real-valued and combinatorial optimization problems. More formally, a RK encoding is a tuple $(\mathbf{x}, dec)$, with $\mathbf{x} \in [0, 1]^l$ and $dec$ the decode function used to transform $\mathbf{x}$ to a permutation $\pi$ in the problem space by sorting $\mathbf{x}$ in non-increasing order along its dimension. Equation 1 shows an example of the decoding process for a 4-dimensional vector $\mathbf{x}$.

$$\begin{aligned} \mathbf{x} &=< 0.7_0, 0.3_1, 0.5_2, 0.2_3 > \\ \pi &= dec(\mathbf{x}) \\ &= sort_{\leq} < 0.7_0, 0.3_1, 0.5_2, 0.2_3 > \\ &=< 3, 1, 2, 0 > \end{aligned} \tag{1}$$

A downside of this approach is the computational overhead introduced by the sorting rquired to map from the RK-space to the permutation space. Even so, key advantage of the RK encoding is that enforces the permutation property on offspring generated via traditional crossover techniques in many EAs. This circumvents the problem of repairing or discarding invalid (non-permutation) offspring in direct encodings. However, the search space of RK encodings suffers from some redundancies: since the final permuation is decided by the order of the random key values, many different RK encodings may mapped to the same solution [8]. As a result, the search algorithm might spend much time on the RK search space without introducing new solutions.

To mitigate this problem, n-ball RK encoding and reduced RK encoding are proposed [8]. The former constrains the n-dimensional search space into a region of an n-ball to reduce the volume of the search space by dividing each solution that falls outside of this space by its norm and adding Gaussian noise. The latter reduces the dimensionality of the search space to $n - 1$ by deriving the position the the $n^{th}$ dimension by the negative sum of all other dimensions.

## 2.3 Differential evolution

Following the general structure of GAs, DE is a real-valued population-based search algorithm proposed by [15]. DE iteratively improves the solutions in the population using evolutionary operations, including mutation, crossover, and selection. The main difference between DE and a general GA is that the cross over operation is operated on the mutated offsprings.

DE generates new solutions by first linearly combining the selected solutions in the current population, at the same time, DE makes sure that the offsprings are not randomly generated, and the population will gradually move toward regions with better fitness scores. Then, the cross over operation is applied to the mutated offspring and a solution in the current population to decide which genes will be pass on to the next generation. In this work we focus on the standard version of DE, often referred to as `DE/rand/1`. This flavor of DE is governed by two configurable parameters: the crossover probability $c$, and the factor $f$ that influence how much of the original genotype is kept and how much the variation operator influences the dimensions of the solutions subject to change, respectively.

Since DE has few assumptions on the underlying optimization problems, it is an efficient and simple global optimizer for a wide variety of continuous optimization problems. However, the performance of DE is influenced by its control parameters, such as the population size, the crossover operator, the mutation factor, and the selections of base vector and difference vectors. Therefore, DE's control parameters need to be adjusted for its target problem.

# 3 PROPOSED IMPROVEMENTS

To enhance the baseline RK-based DE algorithms, we propose two local search heuristics. We chose these improvements based on the empirical observation of irregular behaviour of the DE evolutionary algorithm for several TSP instances. In particular, we noticed the convergence of the solution was rather slow when parameters did not provide sufficient variance, and unreliable on instances where the DE variation operator did not allow for promising areas of the search space to be consistently explored. We hypothesize that this weakness of the baseline is caused by the incompatibility of the DE operator with the TSP problem structure: changes in the real-valued search space may be too small to cause a change in the encoding and thus take longer to converge, or may be too strong to maintain good substructures in the population.

To remedy this issue, we turn to local search heuristics to alter the solutions during recombination. LS subroutines used in more complex EAs have been shown to produce state of the art results for many problems whose structures can be effectively exploited by these strategies, including TSP. We hypothesize that this would be especially effective in an RK-based setting because of two main reasons. First, the local search subroutine could introduce the variation required by the algorithm to alter the solution, when mutation in the encoding space is not sufficiently strong. Second, TSP has structural properties that allow solutions altered by certain LS operators

to be evaluated in constant time, which makes LS efficient enough to be used a subroutine as part of the recombination operator.

To this end, we implemented two TSP-specific local search heuristics: the standard 2-opt operator that swaps two edges in the tour to create a new solution and a novel operator which we call 2-node exchange (2-ne) that swaps two nodes in the tour and their adjacent edges.

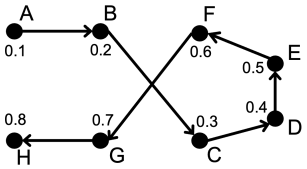



**Figure 1: A RK-based TSP example. The real numbers are the random keys for each city.**

## 3.1 2-opt

The 2-opt local search is originally proposed by [5] to solve TSP problem. The main idea is to find an improvement, a new solution with smaller traveling distance, in the neighborhood by switching two edges. The search is repeated until there is no better solutions in the neighborhood or a local search budget has been met. As above mentioned, the solution evaluation in LS for TSP can be done in constant time using partial evaluation. This is because we only need to calculate the distance changes from those reconnected edges. In 2-opt, only two edges are reconnected, so the new traveling distance can be calculated as

$$distance_{new} = distance_{org} - distance(e_i) - distance(e_j) + distance(e_{i'}) + distance(e_{j'})$$

, where $e_i$ and $e_j$ are the selected edges and $e_{i'}$ and $e_{j'}$ are their reconnected counterparts.

To further demonstrate how to apply 2-opt on a RK-based TSP, an example is present in Figure 1, which shows a route from A to H, whose visiting order is decided by the random key values that are shown next to the vertices.

If $e_{BC}$ and $e_{FG}$ are selected, these two edges are removed and $e_{BF}$ and $e_{CG}$ are built. Knowing this, we can calculate the new traveling distance using partial evaluation, and check if the new distance is smaller than the current best one. If so, the random key vector needs to be modified to create a new route. This can be done by changing the random key values of F and C. However, to make sure all the other connections remain the same, the random keys between F and C also need to be re-ordered as shown in Figure 2. In implementation, we can use the entries of the solution, which is in discrete space, as the index to retrieve the corresponding random keys. Therefore, the random key re-order can be achieved by creating one forward index iterator starting from the first edge's tail, and the other one backward starting from the second edge's head, and then swapping the retrieved random keys until the two iterators meet in the middle. In Figure 2, the random keys of D and C, and E and D are swapped. Note that the visiting order between F and C is changed, but the traveling distance is the same. The 2-opt operation on RK-based TSP can be summarized in Algorithm 1.

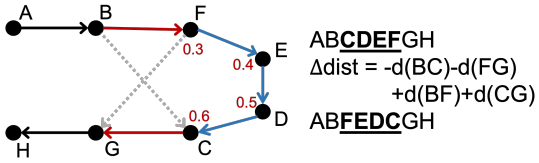



**Figure 2: 2-opt on the RK-based TSP example in Figure 2. The changed random keys are explicitly denoted. Gray: removed edges. Red: new edges. Blue: edges that only change directions.**

**Algorithm 1:** 2-opt operation on RK-based TSP

**Input :** A TSP graph with defined distance on edges $e$,
a route solution $sol$,
the route's RK encoding $rks$,
the route's traveling distance $dist$,
the search budget $B$

**Output :** A route solution $sol$,
the route's RK encoding $rks$,
the route's traveling distance $dist$

$N = 0$
**while** $N < B$ **do**
  Randomly select two nodes, $i$ and $j$, where $0 <= i < j < length(sol)$
  Identify two edges:
  $e0 = e[i-1, i]$, $e1 = e[j, j+1]$
  Reconnect two edges:
  $e0' = e[i-1, j]$, $e1' = e[i, j+1]$
  Partial evaluation:
  $dist_{new} = dist - dist[e0] - dist[e1] + dist[e0'] + dist[e1']$
  **if** $dist_{new} < dist$ **then**
    $dist = dist_{new}$
    Reorder $rks$ between $sol[i]$ and $sol[j]$
    Reverse the order of $sol$ from $i$ to $j$
  $N = N + 1$
**return :** $sol$, $rks$, and $dist$

## 3.2 2-ne

We proposed a novel LS strategy called 2-ne, where fewer constraints are imposed and hence introduce higher variation. In 2-ne, it is the node swapping that triggers the reconnecting of the edges, so one pair of node will reconnect four edges, which are the two nodes' the entering paths and leaving paths.

The example in Figure 1 is used to demonstrate 2-ne on RK-based TSP. First, two nodes are randomly selected, and then the resulting reconnected edges are identified for partial evaluation. In Figure 3, F and C are selected, and the four reconnected edges are identified as shown in the figure. If the new distance is better, the random key of the two selected nodes are swapped. Unlike 2-opt, the visiting order between the selected two nodes is not changed. Only the visiting order of the selected two nodes are swapped. The implementation is similar to 2-opt, the entries of the solution are used as the index to

retrieve the corresponding random keys for swapping. Algorithm 2 summarizes the 2-ne operation on RK-based TSP.

It can be seen that in each iteration, four edges are reconnected, which makes the new solutions from 2-ne differ more compared to those from 2-opt, where only two edges are reconnected. For the same reason, the computation of 2-ne is more expensive than 2-opt because the distance changes are from eight components, four for edge removal and the other four for edge connecting. Thus, the partial evaluation of 2-ne could take twice as much time as 2-opt. Although in 2-ne, there is no need to re-order the random keys, the operation reduction is small compared to the additional effort to calculate the traveling distances.

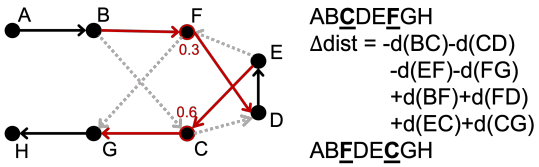


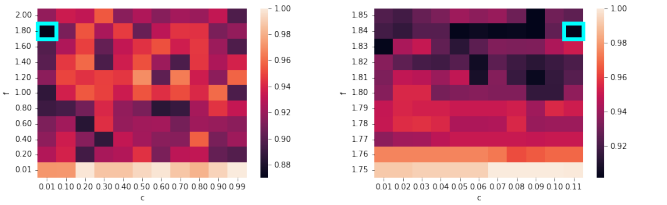

**Figure 3: 2-ne on the RK-based TSP example in Figure 1. The changed random keys are explicitly denoted. Gray: removed edges. Red: new edges.**

**Algorithm 2:** 2-ne operation on RK-based TSP

**Input** : A TSP graph with defined distance on edges $e$,
a route solution $sol$,
the route's RK encoding $rks$,
the route's traveling distance $dist$,
the search budget $B$

**Output**: A route solution $sol$,
the route's RK encoding $rks$,
the route's traveling distance $dist$

$N = 0$
**while** $N < B$ **do**
  Randomly select two nodes, $i$ and $j$, where $0 <= i < j < length(sol)$
  Identify four edges:
  $e0 = e[i-1, i]$, $e1 = e[i, i+1]$, $e2 = e[j-1, j]$, $e3 = e[j, j+1]$
  Reconnect four edges:
  $e0' = e[i-1, j]$, $e1' = e[i, j+1]$, $e2' = e[j-1, i]$, $e3' = e[j, i+1]$
  Partial evaluation:
  $dist_{new} = dist - dist[e0] - dist[e1] - dist[e2] - dist[e3] + dist[e0'] + dist[e1'] + dist[e2'] + dist[e3']$
  **if** $dist_{new} < dist$ **then**
    $dist = dist_{new}$
    Swap $rks[sol[i]]$ and $rks[sol[j]]$
    Swap $sol[i]$ and $sol[j]$
  $N = N + 1$
**return** : $sol$, $rks$, and $dist$



**(a) Heatmap level 1 grid search results for RK on berlin52.**

**(b) Heatmap level 2 grid search results for RK on berlin52.**

**Figure 4: Results of two-level grid search for RK on berlin52.**

## 4 EXPERIMENTS

To gain insight into the effects of LS on the random key encodings, we carried out an empirical study on four instances selected from the TSPLIB dataset [14]. The algorithms were implemented in the Python 3.10 programming languages, and the computational experiments were carried out on a 2014 MacBook Pro running on a Intel Core i7 processor at 2.5 GHz, with 16 GB of RAM. We consider 9 permutations of the encodings and their (non-)LS variants: each of RK, nbRK, and rRK were paired together with no local search, 2-ne based local search (referred to as LS) and 2-opt based local search (referred to as LS2OPT). Subsection 4.1 describes the parameter search procedure executed to answer RQ1. Subsection 4.2 focuses on a qualitative analysis of the best solution found by the algorithms to answer RQ2. Finally, subsection 4.3 analyses the effect of the local search budget on the runtime and solution quality of the algorithm, as put forward in RQ3. To ensure reproducibility, we make the code used for the implementation of both the algorithms and the experiments available as a supplement to this document.

### 4.1 Parameters

To ensure both fairness between the compared algorithms and good performance on the tested instances, the $c$ and $f$ parameters of the DE recombination were tuned using two-level grid search. This procedure was performed on the burma14 and the berlin52 instances. However, because of the fast convergence of the LS-based configurations on smaller problems, only the results for berlin52 were considered the final choice. This problem was chosen as a representative sample from our 4-problem dataset based on its topology and size. We argue that this is a reasonable approximation to make because the goal of this experiment is not to find optimal parameters for each setting, but rather to discover generally good parameters for all instances, in a way that provides insight into the behavior of the algorithm.

For this experiment, the search space was discretized into 11 equidistant points on the scale of $[0.01, 0.99]$ and $[0, 2]$ for the $c$ and $f$ parameters of the standard DE variation operator, respectively. 5 iterations of 10 generations each were performed for each algorithm for all 121 combinations of the two parameters, and the mean value of the final solution was considered as an indicator of the quality of the solution. To further improve the quality of the parameters, the search procedure was extended with a second level. The best parameter settings were selected from the first

**Table 1: Experimental Parameters obtained via two-level grid search.**

| Algorithm | $c$ | $f$ |
|---|---|---|
| RK | 0.11 | 1.84 |
| RKLS | 0.89 | 0.08 |
| RKLS2OPT | 0.91 | 0.08 |
| nbRK | 0.95 | 1.82 |
| nbRKLS | 0.77 | 0.11 |
| nbRKLS2OPT | 0.91 | 0.08 |
| rRK | 0.72 | 0.58 |
| rRKLS | 0.55 | 0.09 |
| rRKLS2OPT | 0.55 | 0.09 |

instance, and the space was discretized again in the square that centers around the best combination. When boundaries were hit, the space was extended in the opposite direction. By repeating the same procedure across all algorithms, fairness in the parameter selection procedure is assured.

A sample of the results of this experiments that focuses on the RK configuration is visualized in Figure 4. The quality of the result is measured proportionately to all other results in the search. Since TSP is a minimization problem, lower results (visualized here with darker colors) are better. Subfigure (a) shows the results for the first level (coarser) grid search, and subfigure (b) displays the results of the second level. The best results for both procedures are highlighted in blue. The results indicate the importance of this experiment in ensuring fairness across all algorithms: at both levels, the mean best solution varies by between 9 and 12 pp. between the best and worst pairs of parameters.

For reproducibility purposes, the final parameters selected by the grid search procedure are given in Table 1. Of note is the level of similarity of the preferences of LS-based algorithms. The $f$ parameter is consistently in the range $[0.08, 0.11]$ for all LS variants, which is a much tighter interval than for the baseline counterparts. These findings support our hypothesis that LS might be most beneficial for less perturbative DE recombination operators. Moreover, the higher values of the $f$ parameter (1.84, 1.82, and 0.58) for the baseline RK encodings also support our hypothesis that the standard version of the algorithm might benefit from recombination that more strongly affects the solutions.

## 4.2 Solution Quality

To assess and compare the local search heuristics for the RK encodings, we collected data on four instances of real-world problems from TSPLIB. These instances are, in increasing size: dantzig42, att48, eil51, and berlin52. We selected these problems because of their topology, which does not show any outstanding pattern that might not be generalizable to most other instances and because their size allow for sufficiently robust experimentation within the constraints of this research. The parameters for these experiments laid out in Table 1. Tournament selection of size 2 was used for all experiments, and each configuration was run 10 independent times. Each run consists of 50 generations.

**Table 2: Statistical significance analysis of the nine algorithms on berlin52 using the Wilcoxon test. The alternative hypothesis of the test was that the row-configuration had a lower objective value for its best solution than the column-configuration. Significant results (with $p < 0.05$ are highlighted in bold text.**

| | RK | RKLS | RKLS 2OPT | nbRK | nbRKLS | nbRKLS 2OPT | rRK | rRKLS | rRKLS 2OPT |
|---|---|---|---|---|---|---|---|---|---|
| RK | - | 1.000 | 1.000 | 0.722 | 1.000 | 1.000 | **0.032** | 1.000 | 1.000 |
| RKLS | **0.001** | - | 1.000 | **0.001** | 0.539 | 1.000 | **0.001** | **0.014** | 1.000 |
| RKLS2OPT | **0.001** | **0.001** | - | **0.001** | **0.001** | 0.216 | **0.001** | **0.001** | **0.010** |
| nbRK | 0.312 | 1.000 | 1.000 | - | 1.000 | 1.000 | **0.001** | 1.000 | 1.000 |
| nbRKLS | **0.001** | 0.500 | 1.000 | **0.001** | - | 1.000 | **0.001** | 0.161 | 1.000 |
| nbRKLS2OPT | **0.001** | **0.001** | 0.812 | **0.001** | **0.001** | - | **0.001** | **0.001** | **0.014** |
| rRK | 0.976 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | - | 1.000 | 1.000 |
| rRKLS | **0.001** | 0.990 | 1.000 | **0.001** | 0.862 | 1.000 | **0.001** | - | 1.000 |
| rRKLS2OPT | **0.001** | **0.001** | 0.993 | **0.001** | **0.001** | 0.990 | **0.001** | **0.001** | - |

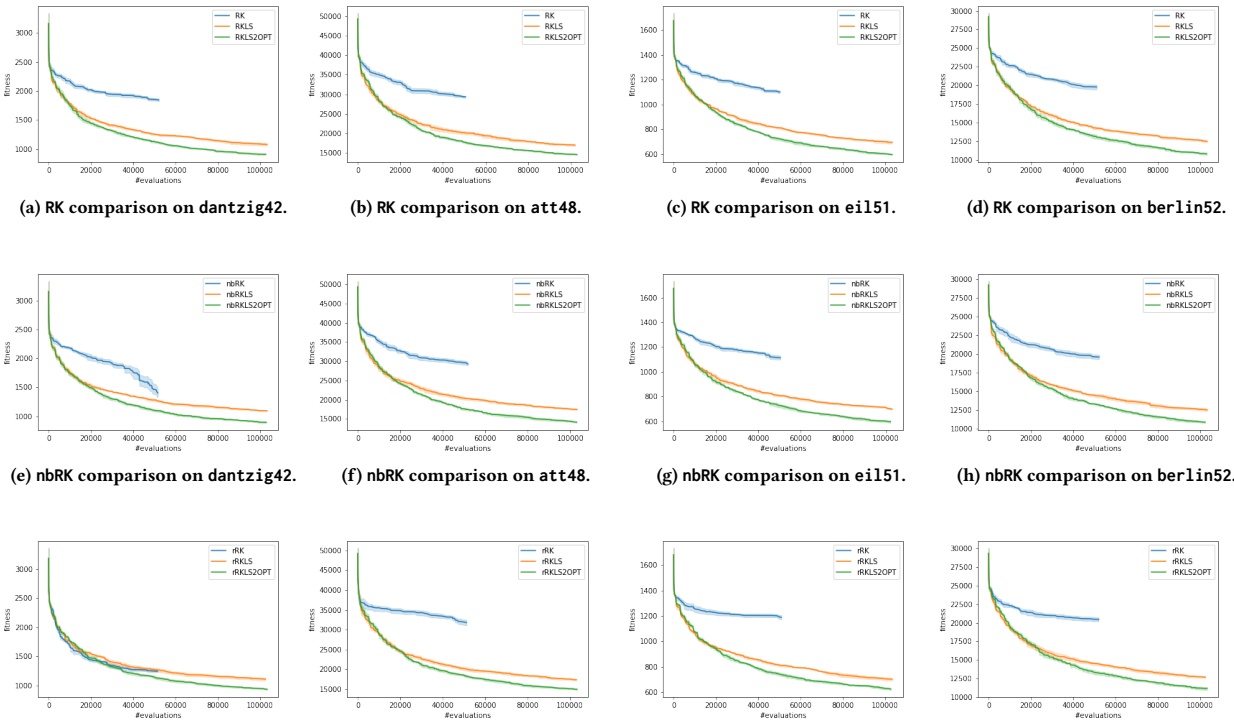
The results of the experiment are highlighted in Figure 5. Each row of subfigures displays the results of a "family" of algorithms on the four instances, and each column contains the results of all algorithms for one of the benchmark problems. A clear trend is visible among almost all configurations. The baseline algorithms perform significantly worse than their LS-enriched counterparts in all but one instance, where all three rRK strategies perform comparably on the dantzig42 problem. The LS heuristic algorithms follow the same pattern of convergence across all 12 comparisons. While the 2-ne version vastly outperforms the baseline, it always falls short of its 2-opt counterpart. However, this trend is only apparent after more than 25 percent of the number of evaluations performed in total: both LS versions perform indistinguishably during the early stages of the search. This indicates that the less aggressive changes introduced by 2-opt might be better suited for finding improvements during the intermediate and late stages of the search algorithm because it tends to check solutions that are "closer" to the original offspring than 2-ne.

Statistical analysis was performed on the results to determine the significance of the differences between the algorithm families. We used the standard Wilcoxon test for this purpose. The test compares two sets of similar samples and computes the $p$-value of the difference between the two being symmetrical around zero. It is a non-parametric version of the paired T-test. We carried out the the pair-wise Wilcoxon test for all algorithms, the results of which are shown in Table 2, for the representative berlin52 problem.

The tests reveal several key insights. First, when considering the baseline algorithms, both RK and nbRK outperform rRK. We believe this is because of the limitations imposed by the rRK encoding on the search space, which constraints the solution's neighbourhood structure. When considering the LS-enriched variants, RKLS2OPT stands out as the best performer, showing significantly better results than all counterparts, except for nbRKLS2OPT, which comes in second. RKLS and nbRKLS are the weakest performers of the LS algorithms, and only show significantly better results than the non-LS baselines.

## 4.3 Effects of LS Search Budget

To analyze the effect of the search budget on the best objective value achieved with the LS versions of the algorithms, we compared the




(a) RK comparison on dantzig42. (b) RK comparison on att48. (c) RK comparison on eil51. (d) RK comparison on berlin52.

(e) nbRK comparison on dantzig42. (f) nbRK comparison on att48. (g) nbRK comparison on eil51. (h) nbRK comparison on berlin52.

(i) rRK comparison on dantzig42. (j) rRK comparison on att48. (k) rRK comparison on eil51. (l) rRK comparison on berlin52.

**Figure 5: Comparison for the three RK-based encodings with the two LS heuristics on four TSPLIB instances.**

6 configurations in terms of fitness of the best solution and average time taken to process one generation. The former metric measures the improvement that an increased budget may bring, while the latter assesses the computational cost incurred to do so. The search budget was varied from 5 to 50 with increments of 5 per run. Each run consists of 20 generations. The experiments were run on the berlin52 instance.

Figure 6 displays the best fitness achieved by each configuration as a function of the search budget. As expected, increasing the search budget significantly increases the quality of the best solution for all algorithms. However, the increase in solution quality differs across the two LS implementation. 2-ne based algorithms perform worse and improve their solutions less than their 2-opt counterparts. This shows further evidence against our hypothesis that more perturbative LS subroutines can increase the performance of the EA. When considering the 2-opt and 2-ne variants in isolation, there is no significant difference between the trends of the three RK-based encodings. The gain is initially superlinear, but the improvement incrementally diminishes as the search budget increases. For both LS configurations, the improvement starts to trend towards sublinearity for budgets of 25 and over.

To compare which version benefits the most from a higher budget, a first degree polynomial was fitted to the results. The slope of the line can than be used to determine which algorithms benefit the most from a higher budget.Table 3 shows that the 2-opt versions of the algorithms have a steeper slope than their non 2-ne counterparts. On average, the 2-opt cofniguration has 0.74 times steeper slope than its 2-ne counterpart, suggesting that 2-opt scales better with a higher LS budget.

Figure 7 contains a visualization of the average cost of a generation as a function of the search budget. The results indicate that all variants scale linearly with the search budget. As hypothesized in subsection 3.2, 2-ne based algorithms are have a higher computational cost than the standard 2-opt. We believe that this is caused by the additional overhead required by performing memory lookups for the costs of more edges at each LS iteration. The checking of several edge cases (such as swapping two nodes adjacent nodes) may also impact the runtime performance of the 2-ne variants. When considering the different encodings, rRK stands out as the most expensive for both LS heuristics, while the difference between RK and nbRK is negligeable. This shows that the additional step of computing the summation of the $l-1$ dimensions of the encoding is a significant overhead and should be considered when choosing an encoding.

## 5 DICUSSION AND CONCLUSIONS

In this study, we investigated the performance of RKLS-based DE algorithms for TSP. Three different RK encoding methods, RK, nbRK, and rRK are considered to explore the impact of using RKs with less search space redundancy. To combat the observed inefficiency of using DE for TSP, two LS heuristics, 2-opt and 2-ne, are applied.

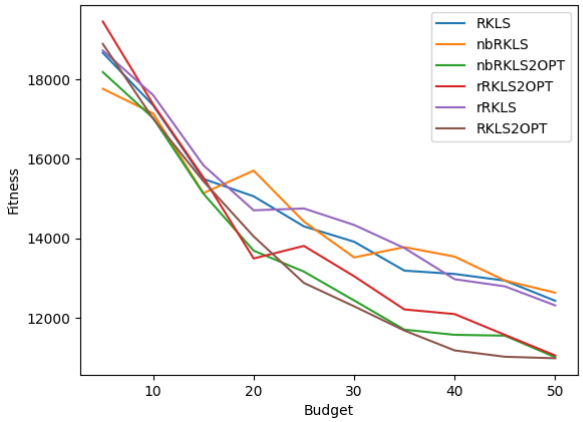



**Figure 6: Fitness of the algorithms as a function of budget, on `berlin52`.**

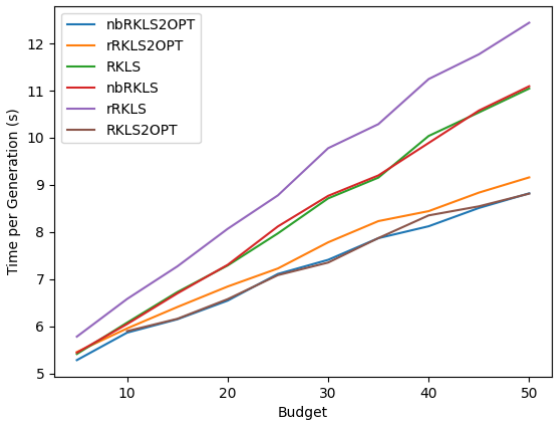



**Figure 7: Runtime of the algorithms as a function of LS budget, compared on `berlin52`.**

**Table 3: Estimated slope and intercept for fitness versus budget data.**

| Algorithm | Slope | Intercept |
|---|---|---|
| RKLS | -127.1 | 18143 |
| rRKLS | -131.9 | 18411 |
| nbRKLS | -109.3 | 17666 |
| RKLS2OPT | -172.0 | 18277 |
| nbRKLS2OPT | -154.1 | 17790 |
| rRKLS2OPT | -167.1 | 18563 |

Our findings after analyzing the results from experiments focusing on different aspects are described as below.

First, under the non-LS setting, the performance of RK and nbRK are similar, while rRK performs the worst. However, in LS-based algorithms, using nbRK and rRK does not result in better performance. We suspect that the redundancy reduction matters more when the search space is much larger. Thus, the improvement from nbRK and rRK might be more significant in instances on a larger scale.

Second, LS heuristics are capable of introducing informative solution variations to TSP. This can be found from the results of optimal parameter search, where LS-based algorithms prefer less perturbative DE operators while non-LS-based algorithms benefit from a higher degree of re-combinations. The solution quality experiment also shows that LS-based methods can evolve toward a better search space while the standard algorithms fail to reach a better search region after almost half of the lifetime of LS-based algorithms.

Third, compared to `2-ne`, `2-opt` can move toward a better search region faster. This is because the higher variation of the solutions found by `2-ne` could prevent the search algorithm from finding a better solution within a smaller neighborhood, which is especially crucial in the later stage since the algorithm is meant to exploit the existing solutions as much as possible.

Last, the search budget is positively correlated to both the runtime per generation and the performance of LS-based algorithms. Between the two LS heuristics, increasing the search budget introduces more improvements to `2-opt` implementations in less additional runtime. This result aligns with those found in the solution quality experiment because the higher variation of `2-ne` prevents it from exploring a better search region when search is about to converge. As such, it cannot significantly increase its performance over `2-opt` more even with more search budget. Finally, the results show no specific trends in the algorithm improvements for different RK encodings.

In the future, several research directions can be explored to imrpove RK-based encodings using LS. One option is the development of a RK-specific LS heuristic that perturbs the solution with regard to the RK-space rather than the solution space. A specilized continuous-space heuristic might prove beneficial by directly countering the shortcomings of DE in the TSP setting, without alternating the variation between the two domains. In addition, the development of a TSP-specific RK variant that exploits problem-specific knowledge (such that smaller variation like that introduced like `2-opt` is preferable over stronger changes) might also be a promising direction to explore.

In conclusion, running RKLS-based TSP using DE on instance `dantzig42`, `att48`, `eil51`, and `berlin52`, we do not observe significant improvement from different RK encoding methods. With the addition of LS heuristics, the performance is improved significantly since LS can reduce the inefficiency of using DE algorithm for TSP. Furthermore, small but effective variations are important in the later search stage, which explains the better performance discrepancy between `2-opt` over `2-ne`.

# REFERENCES


[1] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2015. Linear ordering optimization with a combinatorial differential evolution. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2135–2140.

[2] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2020. Variable neighborhood algebraic differential evolution: An application to the linear ordering problem with cumulative costs. *Information Sciences* 507 (2020), 37–52.

[3] James C. Bean. 1994. Genetics and Random Keys for Sequencing and Optimization. (1994).

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[5] Georges A Croes. 1958. A method for solving traveling-salesman problems. *Operations research* 6, 6 (1958), 791–812.

[6] Iztok Fister and Janez Brest. 2011. Using differential evolution for the graph coloring. In *2011 IEEE Symposium on Differential Evolution (SDE)*. IEEE, 1–7.

[7] Da Gao, Gai-Ge Wang, and Witold Pedrycz. 2020. Solving fuzzy job-shop scheduling problem using DE algorithm improved by a selection mechanism. *IEEE Transactions on Fuzzy Systems* 28, 12 (2020), 3265–3275.

[8] Pavel Krömer, Vojtěch Uher, and Václav Snášel. 2021. Novel Random Key Encoding Schemes for the Differential Evolution of Permutation Problems. *IEEE Transactions on Evolutionary Computation* 26, 1 (2021), 43–57.

[9] Shadi Mahmoudi and Shahriar Lotfi. 2015. Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem. *Applied soft computing* 33 (2015), 48–64.

[10] Peter Merz and Bernd Freisleben. 1997. Genetic local search for the TSP: New results. In *Proceedings of 1997 Ieee International Conference on Evolutionary Computation (Icec'97)*. IEEE, 159–164.

[11] Nasimul Noman and Hitoshi Iba. 2008. Differential evolution for economic load dispatch problems. *Electric power systems research* 78, 8 (2008), 1322–1331.

[12] Quan-Ke Pan, Ling Wang, and Bin Qian. 2009. A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. *Computers & Operations Research* 36, 8 (2009), 2498–2511.

[13] Millie Pant, Hira Zaheer, Laura Garcia-Hernandez, Ajith Abraham, et al. 2020. Differential Evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence* 90 (2020), 103479.

[14] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.

[15] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

[16] M Fatih Tasgetiren, Ponnuthurai N Suganthan, and Quan-Ke Pan. 2010. An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem. *Appl. Math. Comput.* 215, 9 (2010), 3356–3368.

[17] Nico LJ Ulder, Emile HL Aarts, Hans-Jürgen Bandelt, Peter JM Van Laarhoven, and Erwin Pesch. 1990. Genetic local search algorithms for the traveling salesman problem. In *International Conference on Parallel Problem Solving from Nature*. Springer, 109–116.

[18] Yuan Yuan and Hua Xu. 2013. Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering* 65, 2 (2013), 246–260.