Random is Faster than Systematic in Multi-Objective Local Search

Zimin Liang, Miqing Li*

School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K.
zxl525@student.bham.ac.uk, m.li.8@bham.ac.uk

Abstract

Local search is a fundamental method in operations research and combinatorial optimisation. It has been widely applied to a variety of challenging problems, including multi-objective optimisation where multiple, often conflicting, objectives need to be simultaneously considered. In multi-objective local search algorithms, a common practice is to maintain an archive of all non-dominated solutions found so far, from which the algorithm iteratively samples a solution to explore its neighbourhood. A central issue in this process is how to explore the neighbourhood of a selected solution. In general, there are two main approaches: 1) systematic exploration and 2) random sampling. The former systematically explores the solution's neighbours until a stopping condition is met – for example, when the neighbourhood is exhausted (i.e., the best improvement strategy) or once a better solution is found (i.e., first improvement). In contrast, the latter randomly selects and evaluates only one neighbour of the solution. One may think systematic exploration may be more efficient, as it prevents from revisiting the same neighbours multiple times. In this paper, however, we show that this may not be the case. We first empirically demonstrate that the random sampling method is consistently faster than the systematic exploration method across a range of multi-objective problems. We then give an intuitive explanation for this phenomenon using toy examples, showing that the superior performance of the random sampling method relies on the distribution of "good neighbours". Next, we show that the number of such neighbours follows a certain probability distribution during the search. Lastly, building on this distribution, we provide a theoretical insight for why random sampling is more efficient than systematic exploration, regardless of whether the best improvement or first improvement strategy is used.

Code — https://github.com/Zim-L/AAAI26LS

Supplementary material — https://github.com/Zim-L/AAAI26LS/blob/main/Supplementary.pdf

Introduction

Local search (LS) is a class of search heuristics that move from solution to solution in the space of candidate solutions by applying local changes. LS has been widely used to solve various challenging search and optimisation problems, including SAT (Liang, Zhou, and Yin 2025; Ye, Luo, and Cai 2025; Zheng et al. 2025), max c-cut (Garvardt et al. 2024), max quasi-clique (Chen et al. 2021), max k-plex (Sun et al. 2024), graph edge partition (Guo et al. 2021), Latin square completion (Xie et al. 2024), matroid optimisation (Benabbou et al. 2021), integer linear programming (Lin et al. 2024), and other applied problems (Shatabda, Newton, and Sattar 2013; Zhang et al. 2024; Su et al. 2021; Grüttemeier, Komusiewicz, and Morawietz 2021).

Multi-objective optimisation refers to an optimisation scenario where there is more than one objective to be considered simultaneously. LS is a popular tool in tackling multi-objective combinatorial optimisation problems (MOCOPs) (Blot, Kessaci, and Jourdan 2018), due to its simple algorithmic structure and neighbourhood-driven exploration which fits well with the discrete search space.

Early multi-objective LS methods were often desgined through converting an MOCOP into several single-objective problems and using scalar fitness (aggregated from multiple objectives) to rank solutions. Representative algorithms include (Ishibuchi and Murata 1998; Jaszkiewicz 2002; Paquete and Stützle 2003). Later, Pareto-based local search (PLS) heuristics have received increasing attention and becomes a mainstream method (Paquete, Chiarandini, and Stützle 2004; Giel 2003; Lust and Teghem 2010; Dubois-Lacoste, López-Ibáñez, and Stützle 2015; Drugan and Thierens 2012; Jaszkiewicz 2018; Shi et al. 2022; Ren et al. 2023; Kang et al. 2024). PLS adopts Pareto dominance to compare solutions (Paquete, Chiarandini, and Stützle 2004). It maintains an archive of non-dominated solutions found so far and iteratively samples solutions from the archive to explore their neighbourhoods. Such PLS algorithms show competitive performance on MOCOPs such as the multi-objective TSP (Lust and Teghem 2010), knapsack (Alsheddy and Tsang 2010), NK-landscape (Aguirre and Tanaka 2005) and scheduling (Liefooghe et al. 2012) problems. More recently, several studies (Inja et al. 2014; Phan and Luong 2023; Li et al. 2024) have shown the performance advantage of PLS over well-established multi-objective evolutionary algorithms (e.g., NSGA-II (Deb et al. 2002), SMS-EMOA (Beume, Naujoks, and Emmerich 2007) and MOEA/D (Zhang and Li 2007)).

A key issue in PLS algorithms is how to explore the neigh-

*Corresponding author.

bourhood of a solution sampled from the archive. In general, there are two approaches: 1) systematic exploration and 2) random sampling. The former, denoted by *systematic* PLS or *s*-PLS, systematically explores the solution's neighbours until a stopping condition is met – for example when the neighbourhood is exhausted (i.e., the best improvement strategy), or once a better solution is found (i.e., the first improvement stragtegy). The latter, denoted by *randomised* local search or *r*-PLS, randomly selects and evaluates only one neighbour of the solution. Systematic PLS is popular in the fields of operations research and metaheuristics (Dubois-Lacoste, López-Ibáñez, and Stützle 2015; Derbel et al. 2016; Shi, Zhang, and Sun 2020; Santos et al. 2022; Ceschia, Di Gaspero, and Schaerf 2023; Gao, Liu, and Chen 2025). On the other hand, a few studies consider randomised PLS (Chicano, Whitley, and Tinos 2016; Jaszkiewicz 2018), particularly in the theory community of evolutionary computation (Laumanns, Thiele, and Zitzler 2004; Doerr and Zheng 2021; Dinot et al. 2023; Wietheger and Doerr 2024).

A natural question that arises is which of these two approaches is more efficient. Systematic search might appear preferable, as it methodically explores a solution's neighbourhood and ensures each neighbour is visited at most once. In contrast, randomised PLS may revisit the same neighbours multiple times (as, at each step, it randomly selects a solution from the archive and then randomly picks one of its neighbours), resulting in redundant evaluations.

In this paper, however, we show that randomised PLS is in general faster than systematic PLS. We first empirically demonstrate consistent performance advantage of randomised PLS over systematic PLS across a wide range of problems, no matter whether the best improvement or first improvement strategy of neighbourhood exploration is used. We then give an intuitive explanation for this phenomenon using toy examples. Lastly, we provide theoretical understanding through analysing the runtime of two PLS approaches in finding a good solution (i.e., one that is not dominated by the archive, thus leading to improvement of the archive quality) under the distribution of neighbouring solutions of the considered problems. Key contributions of this work can be summarised as:

- Out of the two major classes of local search algorithms in multi-objective optimisation, we show one is faster than the other, which refutes a commonly-held belief that the two LS algorithm classes may have their own strengths and suit different problems.
- We empirically investigate the neighbourhood structure (the number of good neighbours) during the search process of the PLS algorithms, and demonstrate that they follow a certain probability distribution.
- We theoretically investigate the reason behind our empirical observations, and prove that one algorithm class requires less time to find a good solution under that probability distribution.

It is worth noting that randomised PLS heuristics, such as SEMO (Laumanns, Thiele, and Zitzler 2004), have been frequently studied in the runtime analysis literature within the evolutionary computation theory community (Bian, Qian, and Tang 2018; Doerr and Zheng 2021; Dinot et al. 2023; Zheng et al. 2024; Wietheger and Doerr 2024). These works, along with others in the theory field, focus on the runtime required to find Pareto optimal solutions, offering insights into questions such as whether the algorithm struggles to find a Pareto optimum, and how easy it is to find other optimal solutions in the Pareto front once a single one has been found. Our work does not consider the time required to find optimal solutions. The results presented in this paper does not imply any guarantees of optimality for the solutions found by PLS, but rather how fast different PLS algorithms find a better solution during the search process. This has practical significance, as optimal solutions are often infeasible to obtain within a reasonable time for most real-world MOCOPs (Figueira et al. 2017), and it is valuable to know which types of algorithms can make faster progress.

Preliminaries

Let $\boldsymbol{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ denote an m-objective (minimisation) problem over a decision space X, $\mathbf{x} \in X$, $f_i : X \to \mathbb{R}$. We say $\mathbf{x}$ weakly Pareto-dominates $\mathbf{y}$ (notation: $\mathbf{x} \preceq \mathbf{y}$) iff $\forall\, i \in \{1, \ldots, m\} : \ f_i(\mathbf{x}) \leq f_i(\mathbf{y})$, and $\mathbf{x}$ Pareto-dominates $\mathbf{y}$ (notation: $\mathbf{x} \prec \mathbf{y}$) if at least one inequality is strict. The Pareto set consists of all non-dominated solutions, denoted by $PS = \{\mathbf{x} \in X \mid \nexists\, \mathbf{y} \in X : \mathbf{y} \prec \mathbf{x}\}$; and the Pareto front consists of their corresponding point in the objective space, denoted by $PF = \{\boldsymbol{f}(\mathbf{x}) \mid \mathbf{x} \in PS\}$.

A central concept in local search algorithms is *neighbourhood*, which defines the structure of the search landscape. The neighbourhood of a solution $\mathbf{x}$ is denoted by $\mathcal{N}(\mathbf{x}) \subseteq X \setminus \mathbf{x}$, and typically consists of solutions reachable from $\mathbf{x}$ by a single elementary operation (e.g., a one-bit flip in binary encoded problems).

Two Classes of Pareto-based Local Search

In this work, we focus on Pareto-based local search (PLS) algorithms (Paquete, Chiarandini, and Stützle 2004), which encompass the majority of multi-objective local search heuristics (Blot, Kessaci, and Jourdan 2018). PLS is an iterative search process that performs local variation to solutions and compare them with new solutions using Pareto dominance. It maintains an archive to store all non-dominated solutions found so far. For each iteration, PLS considers a solution in the archive to generate new solution(s) by exploring its neighbourhood. Based on how the solution's neighbourhood is explored, there are two classes of PLS algorithms: systematic PLS (*s*-PLS) and randomised PLS (*r*-PLS).

s-PLS algorithms visit the solution's neighbours one-by-one (and test if they can be put into the archive) until a condition is met. The condition can be neighbourhood exhaustion (i.e., best improvement) or finding a satisfactory solution (i.e., first improvement). In multi-objective optimisation, a satisfactory solution can be defined in different ways, but the two common ones are i) a solution that is not dominated by any solution in the archive and ii) a solution that dominates at least one solution in the archive (Liefooghe et al. 2012; Dubois-Lacoste, López-Ibáñez, and Stützle 2015). We consider both in this study. Once a solution's neighbourhood

Algorithm 1: Systematic Pareto Local Search (*s*-PLS)

Require: max_eval (algorithm terminating condition), $\mathcal{N}$ (neighbourhood function, e.g. 1-bit neighbourhood)

$s \leftarrow random\_solution()$
$explored(s) \leftarrow$ FALSE ▷ Mark s as unexplored
$A \leftarrow \{s\}$ ▷ Place s into the archive
$eval \leftarrow 1$
while $eval \leq max\_eval$ **and** $\exists a \in A : explored(a) =$ FALSE **do**
  $s \leftarrow selection(\{a \in A | explored(a) =$ FALSE$\})$ ▷ Select an unexplored solution from A based on certain strategy (e.g., random selection or using an scalarisation function)
  repeat
    $s' \leftarrow next\_neighbour(\mathcal{N}(s))$ ▷ Get the next neighbour from the neighbourhood of s, usually in lexicographic or random order
    if $\nexists a \in A : a \preceq s'$ **then**
      $A \leftarrow A \cup \{s'\} \setminus \{a' \in A \mid s' \prec a'\}$ ▷ Ensure that the archive only contains unique non-dominated solutions
      $explored(s') \leftarrow$ FALSE
    end if
    $eval \leftarrow eval + 1$
  until $stop\_condition()$ ▷ Stop exploring this neighbourhood if a condition is met, e.g., it is exhausted, or a satisfactory solution is found (e.g., it is non-dominated)
  $explored(s) \leftarrow$ TRUE ▷ Mark s as explored
end while
return A

is explored, it is labelled as "explored" and is never revisited. The framework of *s*-PLS is given in Algorithm 1. *s*-PLS is popular in the area. Many existing PLS heuristics fall into this class, such as the original PLS (Paquete, Chiarandini, and Stützle 2004), MORBC (Aguirre and Tanaka 2005) and its variants (Ide, Aguirre, and Tanaka 2024), GPLS (Alsheddy and Tsang 2010), 2PPLS (Lust and Teghem 2012), anytime PLS (Dubois-Lacoste, López-Ibáñez, and Stützle 2015), PPLS/D (Shi et al. 2022), LOMONAS (Phan and Luong 2023), and a new enhanced PLS (Kang et al. 2024).

Unlike *s*-PLS, for each iteration *r*-PLS randomly selects a solution from the archive and then visit one of its neighbours at random, without tracking which solution has been visited or explored. This purely randomised sampling approach may visit the same neighbour of a solution multiple times, resulting in redundant cost. Algorithm 2 gives the framework of *r*-PLS. Existing work on developing *r*-PLS algorithms is relatively rare. Notable examples include SEMO (Giel 2003; Laumanns, Thiele, and Zitzler 2004), SPLS (Drugan and Thierens 2012) and MPLS (Jaszkiewicz 2018). That said, *r*-PLS has been frequently studied in the evolutionary computation theory community (Bian, Qian, and Tang 2018; Doerr and Zheng 2021; Dinot et al. 2023; Zheng et al. 2024; Wietheger and Doerr 2024), due to its appealing nature (simplicity, minimalism and generalisability).

A relevant question is which of the two classes of PLS is faster. One might assume that *s*-PLS to be more efficient, as it conducts a systematic search and avoids repeatedly exploring the same solutions, particularly when using the first improvement strategy. However, in the following sections, we show that this may not be the case.

Algorithm 2: Randomised Pareto Local Search (*r*-PLS)

Require: max_eval (algorithm terminating condition), $\mathcal{N}$ (neighbourhood function, e.g. 1-bit neighbourhood)

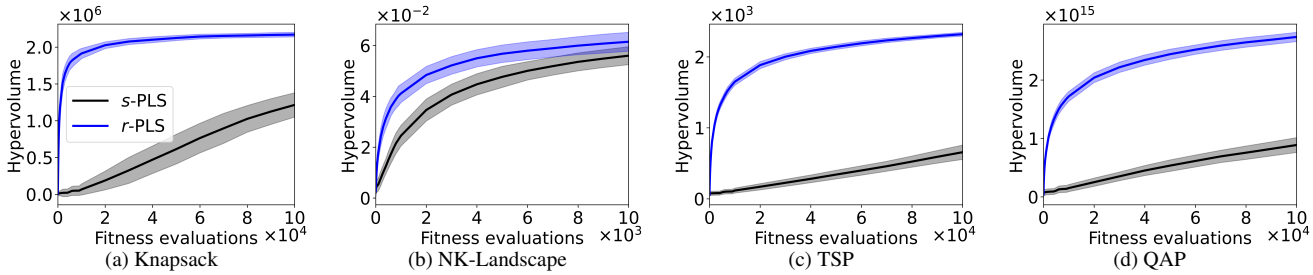
$s \leftarrow random\_solution()$
$A \leftarrow \{s\}$ ▷ Place s into the archive
$eval \leftarrow 1$
while $eval \leq max\_eval$ **do**
  $s \leftarrow selection(A)$ ▷ Select a solution from A based on certain strategy (e.g., random selection or using an indicator/scalarisation function)
  $s' \overset{U}{\sim} \mathcal{N}(s)$ ▷ Sample a neighbour uniformly at random from the neighbourhood of s
  if $\nexists a \in A : a \preceq s'$ **then**
    $A \leftarrow A \cup \{s'\} \setminus \{a' \in A \mid s' \prec a'\}$ ▷ Ensure that the archive only contains unique non-dominated solutions
  end if
  $eval \leftarrow eval + 1$
end while
return A

Empirical Comparison between *s*-PLS and *r*-PLS

We consider the most basic versions of the PLS classes that randomly pick a solution in the archive for neighbourhood exploration, as our focus is on general neighbourhood exploration strategies rather than other tuneable algorithmic components (e.g., selection criteria). That is, for *r*-PLS, we consider SEMO (Laumanns, Thiele, and Zitzler 2004). For *s*-PLS, we consider the canonical version (Paquete, Chiarandini, and Stützle 2004) that randomly selects one unexplored solution from the archive to explore its neighbourhood exhaustively (the best improvement strategy). Additionally, we later consider the first improvement strategy, which was regarded to be faster than the original *s*-PLS (Liefooghe et al. 2012; Dubois-Lacoste, López-Ibáñez, and Stützle 2015).

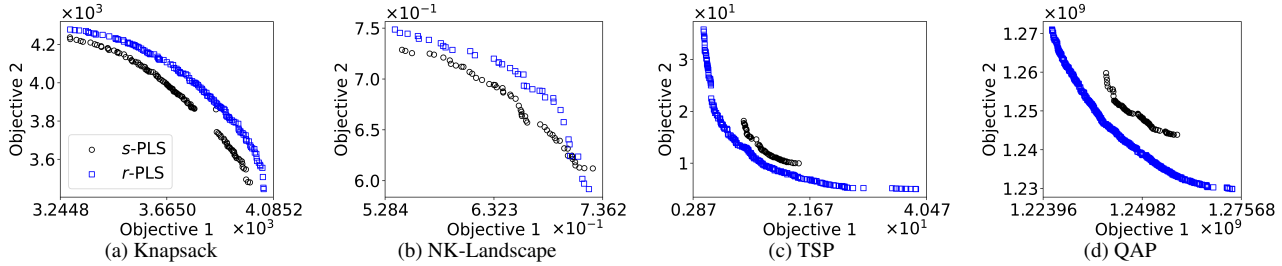
We consider four commonly used MOCOPs, the multi-objective 0/1 knapsack (Teghem 1994), travelling salesman problem (TSP) (Ribeiro et al. 2002), quadratic assignment problem (QAP) (Knowles and Corne 2003) and NK-landscapes (Aguirre and Tanaka 2004). These problems exhibit diverse characteristics, including pseudo-Boolean (knapsack and NK-landscape) and permutation (TSP and QAP) formulations; smooth (knapsack and TSP) to rugged (QAP and NK-landscape with a large K) fitness landscape; order-based (TSP) versus position-based (QAP) permutations; and constrained (knapsack) versus unconstrained (TSP, QAP and NK-landscape) settings. Each problem was instantiated in three sizes (100, 200 and 500 decision variables). We use the hypervolume (HV) (Zitzler and Thiele 1999), a strongly Pareto-compliant indicator capturing convergence, spread, uniformity, and cardinality (Li and Yao 2019), to assess the quality of solutions obtained by search algorithms. For local search operators in *s*-PLS and *r*-PLS, we consider frequently used ones for the corresponding MOCOPs. Due to space limitations, details on the local search operators, problem definitions and settings, and the reference point in HV are given in the supplementary.

Figure 1 shows the average HV (bolded line) and standard



(a) Knapsack (b) NK-Landscape (c) TSP (d) QAP

Figure 1: The hypervolume (HV) trajectory (higher is better) of the *s*-PLS and *r*-PLS across 30 runs on the four MOCOPs with 100 decision variables. The bolded line and shaded area represent the mean and standard deviation of the HV, respectively.



(a) Knapsack (b) NK-Landscape (c) TSP (d) QAP

Figure 2: All non-dominated solutions obtained by the *s*-PLS and *r*-PLS in a typical run on the four MOCOPs, where the Knapsack and NK-landscape are maximisation problems, and the TSP and QAP are minimisation problems.

deviation (shaded area) of *s*-PLS (black) and *r*-PLS (blue) across 30 independent runs on the four problems with 100 decision variables. As can be seen, *r*-PLS obtains a better HV than *s*-PLS throughout the search on all the problems. On the NK-landscape problem, the performance advantage of *r*-PLS is small, while on the other three problems, *r*-PLS demonstrates a clear edge since the very beginning of the search. The difference is even clearer when a larger problem size is involved (see results on the four 200D&500D problems in the supplementary). Figure 2 gives the non-dominated solutions obtained by the two algorithms in a typical run on the four problems when the search ends in Figure 1. It is clear that the *r*-PLS's solutions have better convergence on the Knapsack and NK-landscape problems, and better convergence and diversity on the TSP and QAP.

One explanation for *s*-PLS's slow progress is its exhaustive neighbourhood exploration (the best improvement strategy) – it always evaluates the entire neighbourhood of a solution before moving on. In contrast, the first improvement strategy – the exploration stops once a satisfactory solution is found – can be faster, as shown in (Liefooghe et al. 2012; Dubois-Lacoste, López-Ibáñez, and Stützle 2015). We thus now consider two representative *s*-PLS alternatives that stop the neighbourhood exploration earlier (Dubois-Lacoste, López-Ibáñez, and Stützle 2015). The first one stops the neighbourhood exploration once a non-dominated solution (i.e., it is not dominated by any solution in the archive) is found, denoted by *s*-PLS$_{\not\prec}$. The other one stops once a dominating solution (i.e., it dominates at least one solution in the archive) is found, denoted by *s*-PLS$_{\prec}$.

Figure 3 shows the average hypervolume (bolded line) and standard deviation (shaded area) of the two *s*-PLS algorithms, as well as the basic *s*-PLS and *r*-PLS across 30 independent runs on the 100D four problems. The results on the higher-dimensional problems (200D and 500D) are given in the supplementary. As shown, the two first-improvement *s*-PLS algorithms achieve noticeably better hypervolume performance than the basic *s*-PLS on most problems. Between them, *s*-PLS$_{\prec}$ is faster than *s*-PLS$_{\not\prec}$. However, they are still outperformed by *r*-PLS to a large extent.

The above result is interesting – *r*-PLS, as a simple randomised local search heuristic that may revisit solutions multiple times, has been shown to be substantially more effective than the *s*-PLS algorithms which conduct a systematic search and avoid repeated exploration and revisiting of solutions. Next, we will investigate why this happens.

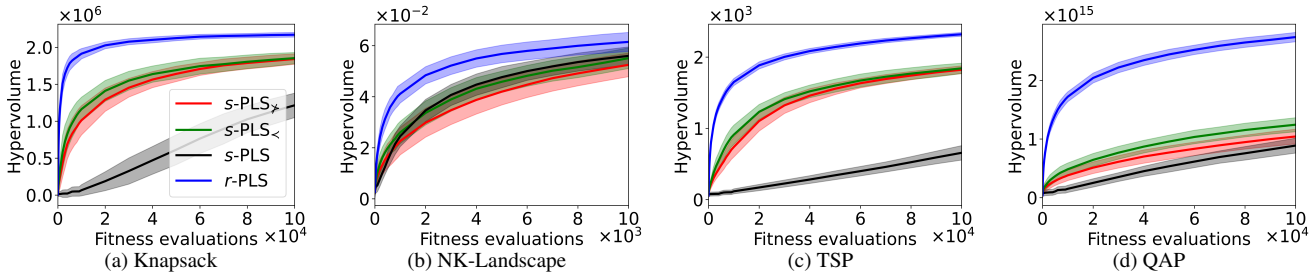
Intuitions from Toy Examples

In this section, we present two toy examples of solution distributions in the archive to help illustrate why and when *r*-PLS can be faster than *s*-PLS. We here are interested in how quickly (in terms of the number of evaluations) an algorithm can find a "good" solution. A good solution is one that is not dominated by the archive, leading to an update of the archive and an improvement of the archive quality.

Example 1

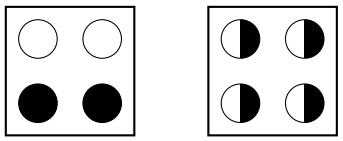
Let us consider a scenario where the archive contains four solutions: half of them *promising* and the other half *unpromising*, as illustrated in Figure 4(a). Here, a promising solution (filled circle in the figure) is one for which every neighbour is a good solution (i.e., non-dominated to the archive); an unpromising solution (empty circle) is one for which every neighbour is dominated by at least one solution in the archive. To start with, both *s*-PLS and *r*-PLS select one solution uniformly at random from the archive to explore.

For *r*-PLS, if a promising solution is selected, its randomly sampled neighbour will be a non-dominated solution, and the search succeeds in a single evaluation. If an unpromising solution is selected, its randomly sampled neigh-



(a) Knapsack (b) NK-Landscape (c) TSP (d) QAP

Figure 3: The hypervolume (HV) trajectory of the *s*-PLS, *s*-PLS$_{\not\prec}$ (it stops when finding a new non-dominated solution), *s*-PLS$_{\prec}$ (it stops when finding a solution dominating any solution in the archive), and *r*-PLS across 30 runs on the four MOCOPs with 100 variables. The bolded line and shaded area represent the mean and standard deviation of the HV, respectively.



(a) Example 1 (b) Example 2

Figure 4: Two toy examples of the distribution of *promising* solutions in the archive. Here, a promising solution (filled circle) means all of its neighbours being non-dominated to the archive; an unpromising solution (empty circle) means all of its neighbours being dominated by at least one solution in the archive; and a half-promising solution (half-filled circle) means half of its neighbours being non-dominated to the archive. In Example 1, the archive contains two promising solutions and two unpromising solutions. In Example 2, all the four solutions in the archive are half-promising.

bour will be a dominated solution; one evaluation is spent and the algorithm proceeds. Since the probability of selecting a promising solution is $1/2$, the expected number of evaluations needed to find a good neighbour is 2.

For *s*-PLS, if a promising solution is selected, the first neighbour picked must be a non-dominated solution, and the search succeeds in a single evaluation. However, if an unpromising solution is selected, *s*-PLS will exhaustively evaluate its entire neighbourhood and waste $|\mathcal{N}|$ evaluations before marking it as explored and selecting another solution for exploration, where $|\mathcal{N}|$ denotes the neighbourhood size. Therefore, while both algorithms have a 50% chance of selecting a promising solution, the potential cost after selecting an unpromising solution is much higher for *s*-PLS. In fact, the expected number of evaluations needed for *s*-PLS to find a good solution in this scenario is $\frac{n|\mathcal{N}|}{n+2}+1$, where n is the number of solutions in the archive. The proof can be found in the supplementary. As can be calculated, for a problem whose neighbourhood size is 10, the expected number of evaluations needed for *s*-PLS to find a good solution in our example is $\frac{4\times 10}{4+2}+1=7.67$, substantially larger than the expected number of evaluations for *r*-PLS (i.e., 2).

Example 2

The above example illustrates an extremely uneven distribution of their neighbours: half of the solutions have only good neighbours, while the other half have only poor ones. In Example 2, we present the opposite case: each solution has an equal mix of good and poor neighbours (Figure 4(b)).

For *r*-PLS, the expected number of evaluations to find a new good solution remains 2, as the probability of selecting a good neighbour for each solution in the archive is $1/2$. For *s*-PLS, since all the solutions have the same distribution that half of their neighbours are good, the probability of selecting a good neighbour for a solution is $1/2$. For the case that *s*-PLS first chooses a poor neighbour, the probability of selecting a good neighbour of that solution at the second pick is $\frac{|\mathcal{N}|}{2(|\mathcal{N}|-1)}$, as the algorithm does not repeatedly visit the same neighbour. This process can continue until there are only good neighbours left for that solution. In fact, the expected number of evaluations needed for *s*-PLS to find a good neighbour is $\frac{|\mathcal{N}|+1}{(|\mathcal{N}|/2)+1}$, which is always less than that of *r*-PLS (i.e., 2). The proof is given in the supplementary.

From the above two examples, we can see the behaviour of the two PLS algorithms depends on the distribution of neighbours of solutions in the archive. If the distribution of solutions' neighbours (in terms of their quality) is even, then *s*-PLS is faster; if the distribution is not even, then *r*-PLS has the edge. But, what is the distribution of the neighbours during the search of the PLS algorithms on common MOCOPs? In the next section, we aim to answer this question.

Distribution of Solutions' Neighbours in *s*-PLS and *r*-PLS

In this section, we investigate the distribution of solutions' neighbours during the search process of *s*-PLS and *r*-PLS on the MOCOPs. Specifically, we count the number of good neighbours among solutions in the archive, and test if this number follows a known discrete probabilistic model (e.g., uniform, binomial and Poisson distributions). Note that this practice is a common approach to test whether an empirical dataset fits a known discrete distribution, for example to fit the number of web requests (which has been found to follow a Zipf distribution) (Breslau et al. 1999), and the citation counts of scientific papers (Xie et al. 2016) (which follows a geometric distribution). In this study, we consider commonly seen discrete distributions (Johnson, Kotz, and Kemp 2005), characterised by their different probabilistic tail type: *uniform* (a balance distribution), *Zipf* (heavy-tailed with polynomial decay), *geometric* (light-tailed with exponential decay), *Poisson* (light-tailed with super-exponential decay), and *binomial* (light-tailed with bounded support and

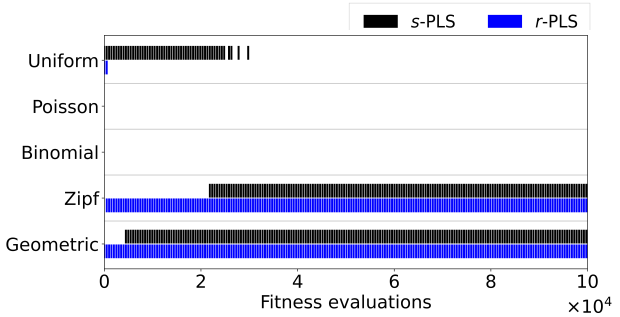


Figure 5: Goodness-of-fit of the distributions with respect to the number of good neighbours among solutions in the archive during the search process of *s*-PLS (black) and *r*-PLS (blue) on the TSP (100 cities). A coloured tick in a row indicates that the algorithm's data at that point was not rejected under the model. For example, for the geometric distribution model, the blue ticks virtually cover the whole band. This indicates that the number of good neighbours among solutions in the archive for *r*-PLS follows a geometric distribution throughout the search process.

a hard cut-off) distributions. Following the common practice, goodness-of-fit here is examined via a χ^2 test at the $\alpha = 5\%$ significance level. Details on distribution fitting, comparison of different distributions, and parameter settings can be seen in the supplementary.

Figure 5 plots the goodness-of-fit of the five discrete distributions with respect to the number of good neighbours among solutions in the archive during the search process of *s*-PLS and *r*-PLS on the TSP. Results on the other three problems can be seen in the supplementary. In the figures, each horizontal band corresponds to a candidate distribution, and at each sampled evaluation, two coloured ticks, black and blue, respectively indicate a good fit for *s*-PLS and *r*-PLS at $\alpha = 0.05$, namely, the χ^2 test does not reject the distribution. As shown in Figure 5, *s*-PLS initially follows a uniform distribution, but for most of its search process, it follows a geometric distribution. In contrast, *r*-PLS exhibits behaviour consistent with both Zipf and geometric distributions. Note that multiple distributions can provide statistically acceptable fits. To further compare Zipf and geometric, we use Akaike's Information Criterion (AIC) (Akaike 1974), which considers both the likelihood of the fit and model complexity (i.e., it penalises distributions with more free parameters). Among the two, the geometric distribution consistently yields a lower AIC, indicating a better fit (see the supplementary for details). In summary, the geometric distribution best describes the behaviour of both *s*-PLS and *r*-PLS on all the considered problems.

The geometric distribution is defined by the probability mass function $\mathbb{P}(G = k) = p(1-p)^k$, where $k \in \mathbb{N}$ and $p \in (0, 1]$. It is light-tailed, meaning that its probability mass decays exponentially as k increases. Unlike uniform distribution (e.g., Example 2 in the previous section) where distribution of the number of good neighbours among solutions is even, the light-tailed yields a highly uneven distribution – most solutions have only a few good neighbours, while those with many good neighbours exist but are rare.

Theoretical Analysis

The above result demonstrates that geometric distribution model is the fittest one for the two PLS algorithms during their search process. In this section, building on the previous finding, we aim to provide theoretical explanation for why *r*-PLS is more efficient than *s*-PLS. That is, we analytically compare the expected time of the two algorithms to find a new good neighbour, under the same geometric distribution (i.e., the same parameter p value) that the number of good neighbours among the solutions in the archives follows. Note that despite both algorithms fitting the same distribution model, they are likely to follow different geometric distributions (i.e., different p values) at a specific timestep. That said, considering the same distribution allows us to analyse their efficiency at an identical search state, hence the performance difference can be attributed solely to the algorithms' neighbourhood exploration approaches, i.e., random sampling versus systematic exploration.

We now begin with a lemma, which will be used in the proof for *s*-PLS in the main theorem.

Lemma 1. *Suppose a set of N ($N \in \mathbb{Z}^+$) solutions contains k good ones ($1 \leq k \leq N$). Assume the solutions in the set are visited one by one in a random order until a good solution is found. Let J denote the number of solutions visited. Then, the expected value of J is $\mathbb{E}[J] = \frac{N+1}{k+1}$.*

Proof. This process is equivalent to sampling uniformly at random without replacement from N items, among which k items are labelled. The number of items examined until encountering the first labelled one is known to have expected value $\mathbb{E}[J] = \frac{N+1}{k+1}$ (Knuth 1997, Sec. 3.4.1). □

This lemma indicates that to first find a good neighbour of a solution, the expected number of its neighbours that *s*-PLS needs to visit is $\frac{|\mathcal{N}|+1}{k+1}$, where $\mathcal{N}$ denotes the neighbourhood size of the problem and k denotes the number of good neighbours the solution has.

Theorem 1 (*r*-PLS is faster than *s*-PLS in finding the next good solution)**.** *Let the number of good neighbours among solutions in the archive of* s*-PLS and* r*-PLS follows a geometric distribution with parameter $p \in (0, 1]$, i.e., $\mathbb{P}(G = k) = p(1-p)^k,\ k \in \mathbb{N}$. Then, for any neighbourhood size $|\mathcal{N}| \in \mathbb{Z}^+$,* r*-PLS has less expected time (fewer evaluations) than* s*-PLS in finding the next good solution.*

Proof. For *r*-PLS, at each iteration, a random neighbour of a randomly selected solution from the archive is sampled. Here, the probability of finding a good neighbour is given by $p_{\text{success}} = \frac{\mathbb{E}[G]}{|\mathcal{N}|}$, where $\mathbb{E}[G]$ represents the expected number of good neighbours. Since G follows a geometric distribution, $\mathbb{E}[G] = \frac{1-p}{p}$ (Johnson, Kotz, and Kemp 2005). Thus, the expected number of evaluations for *r*-PLS to find the next good neighbour is:

$$\mathbb{E}[T_{r\text{-PLS}}] = \frac{1}{p_{\text{success}}} = \frac{|\mathcal{N}|}{\mathbb{E}[G]} = \frac{p|\mathcal{N}|}{1-p}. \tag{1}$$

Now let us consider *s*-PLS. We first compute the expected number of evaluations required when exploring a single solution from the archive. There are two possible cases. The

first case is that the solution has no good neighbour; this occurs with probability $\mathbb{P}(G=0) = p(1-p)^0 = p$ and requires the algorithm to scan the entire neighbourhood, so the expected number of evaluations is $p|\mathcal{N}|$. The second case is that the solution contains $k \geq 1$ good neighbours; this occurs with probability $\mathbb{P}(G=k) = p(1-p)^k$ and it requires the algorithm to visit $\frac{|\mathcal{N}|+1}{k+1}$ neighbours on average (Lemma 1), so the expected evaluation number is $p(1-p)^k \cdot \frac{|\mathcal{N}|+1}{k+1}$ Thus, the overall expected number of evaluations to explore a solution is as follows.

$$\begin{aligned}
\mathbb{E}[T^*] &= p|\mathcal{N}| + \sum_{k=1}^{|\mathcal{N}|} p(1-p)^k \cdot \frac{|\mathcal{N}|+1}{k+1} \\
&= p|\mathcal{N}| + p(|\mathcal{N}|+1) \sum_{j=2}^{|\mathcal{N}|+1} \frac{(1-p)^{j-1}}{j} \quad (\text{let } j = k+1) \\
&= p|\mathcal{N}| + p(|\mathcal{N}|+1) \cdot \frac{1}{1-p}\Big(\sum_{j=1}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j} - (1-p)\Big) \\
&= \frac{p|\mathcal{N}|+1}{1-p} \sum_{j=1}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j} - p
\end{aligned}$$

Now that we have the expected number of evaluations required to explore a single solution, the total expected number of evaluations is obtained by multiplying this value by the expected number of solutions explored from the archive. Since the probability that a randomly selected solution has at least one good neighbour is $1 - P(G=0) = 1-p$, the expected number of solutions explored is $\frac{1}{1-p}$. Thus, the expected number of evaluations for *s*-PLS to find the next good neighbour is:

$$\mathbb{E}[T_{s\text{-PLS}}] = \frac{\mathbb{E}[T^*]}{1-p} = \frac{p(|\mathcal{N}|+1)}{(1-p)^2} \sum_{j=1}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j} - \frac{p}{1-p}. \quad (2)$$

We now compare the number of evaluations required by *s*-PLS and *r*-PLS via examining their ratio.

$$\begin{aligned}
\frac{\mathbb{E}[T_{\text{s-PLS}}]}{\mathbb{E}[T_{\text{r-PLS}}]} &= \frac{\frac{p(|\mathcal{N}|+1)}{(1-p)^2} \sum_{j=1}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j} - \frac{p}{1-p}}{\frac{p|\mathcal{N}|}{1-p}} \\
&= \frac{|\mathcal{N}|(1-p) + (|\mathcal{N}|+1) \sum_{j=2}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j}}{|\mathcal{N}|(1-p)} \quad (3) \\
&= 1 + \frac{|\mathcal{N}|+1}{|\mathcal{N}|} \cdot \frac{\sum_{j=2}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j}}{1-p} > 1
\end{aligned}$$

□

The above proof shows that *r*-PLS requires fewer expected evaluations than *s*-PLS to find a good solution. This result does not distinguish between the best-improvement and first-improvement strategies within *s*-PLS, as it focuses solely on identifying a good solution the algorithm first encounters.

Further looking at the ratio of the expected evaluations of the two algorithms $\frac{\mathbb{E}[T_{s\text{-PLS}}]}{\mathbb{E}[T_{r\text{-PLS}}]}$, the last part of the second term $\frac{\sum_{j=2}^{|\mathcal{N}|+1} \frac{(1-p)^j}{j}}{1-p}$ can be rewritten as $\sum_{j=2}^{|\mathcal{N}|+1} \frac{(1-p)^{(j-1)}}{j}$. This value increases with increasing $(1-p)$ (decreasing p), indicating that when good neighbours are abundant, the difference between *s*-PLS and *r*-PLS becomes even greater. This result confirms the earlier observations that the performance gap between *r*-PLS and *s*-PLS is more pronounced in the early stages of the search (see Figures 1), when it is easier to find good neighbours around the current solutions.

Noted that in single-objective optimisation, there is a trade-off between randomised and systematic LS, called Best-from-Multiple-Selections (BMS) (Cai 2015). BMS extends the idea of random neighbour sampling by drawing k neighbours of a solution and retaining the best. Under the probabilistic model adopted here, BMS acts as a generalisation of *r*-PLS (corresponds to $k=1$). The expected number of evaluations of BMS becomes $\mathbb{E}[T_{r-\text{PLS}}] \cdot (1 + O(k/|\mathcal{N}|))$ (see the supplementary), slightly worse than that of *r*-PLS.

It is worth pointing out that the theorem presented above is a *time-to-next* comparison. *r*-PLS, under the same distribution of the number of good neighbours, is always faster than *s*-PLS in finding a new good solution. It is not an *end-of-run* analysis and thus does not imply that the final solutions obtained by *r*-PLS – once the search terminates (under a sufficient time) – are necessarily better than those produced by *s*-PLS. In fact, as the search progresses, the neighbourhood structure of solutions continues to change, and good neighbours become increasingly scarce. Since *r*-PLS finds good solutions more quickly, the underlying distribution may shift more rapidly (i.e., an increasing p in the geometric distribution), hence potentially making it more difficult to discover a new good solution. That said, despite that here we cannot tell which algorithm is better in the end, *r*-PLS is much faster than *s*-PLS in the early stages of the search when the distribution is similar, particularly on large-scale problems, where a larger neighbourhood size $|\mathcal{N}|$ leads to a greater ratio (Eq. 3). This is echoed by the results on the 200D and 500D problems (see the supplementary).

Conclusion

This work has demonstrated, both empirically and theoretically, that among two standard multi-objective local search approaches, the randomised variant is more efficient than the systematic one. We thus recommend employing randomised local search in multi-objective optimisation when search efficiency is a priority, especially in large-scale problems.

The paper experimentally found that the number of good neighbours encountered during the PLS search follows a tail distribution (specifically, a geometric distribution) rather than being uniform or Gaussian-like. This observation aligned with the general pattern in many optimisation problems, where higher-quality solutions are rarer – leading to a decreasing number of good neighbours as the search progresses. However, this may not hold in certain pseudo-Boolean benchmarks (Liang and Li 2025), such as OneMinMax (Giel and Lehre 2006), LOTZ (Laumanns et al. 2002), OJZJ (Doerr and Zheng 2021). In such cases, systematic PLS may be faster.

References

Aguirre, H.; and Tanaka, K. 2005. Random bit climbers on multiobjective MNK-landscapes: effects of memory and population climbing. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 88(1): 334–345.

Aguirre, H. E.; and Tanaka, K. 2004. Effects of elitism and population climbing on multiobjective MNK-landscapes. In *CEC*, 449–456.

Akaike, H. 1974. A new look at the statistical model identification. *IEEE Trans. Autom. Control.*, 19(6): 716–723.

Alsheddy, A.; and Tsang, E. E. P. K. 2010. Guided Pareto local search based frameworks for biobjective optimization. In *CEC*, 1–8.

Benabbou, N.; Leroy, C.; Lust, T.; and Perny, P. 2021. Combining preference elicitation with local search and greedy search for matroid optimization. In *AAAI*, 12233–12240.

Beume, N.; Naujoks, B.; and Emmerich, M. 2007. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *Eur. J. Oper. Res.*, (3): 1653–1669.

Bian, C.; Qian, C.; and Tang, K. 2018. A general approach to running time analysis of multi-objective evolutionary algorithms. In *IJCAI*, 1405–1411.

Blot, A.; Kessaci, M.-É.; and Jourdan, L. 2018. Survey and unification of local search techniques in metaheuristics for multi-objective combinatorial optimisation. *Journal of Heuristics*, 24(6): 853–877.

Breslau, L.; Cao, P.; Fan, L.; Phillips, G.; and Shenker, S. 1999. Web caching and Zipf-like distributions: evidence and implications. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, 126–134.

Cai, S. 2015. Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In *AAAI*, 747–753.

Ceschia, S.; Di Gaspero, L.; and Schaerf, A. 2023. Educational timetabling: Problems, benchmarks, and state-of-the-art results. *Eur. J. Oper. Res.*, 308(1): 1–18.

Chen, J.; Cai, S.; Pan, S.; Wang, Y.; Lin, Q.; Zhao, M.; and Yin, M. 2021. NuQClq: An effective local search algorithm for maximum quasi-clique problem. In *AAAI*, 12258–12266.

Chicano, F.; Whitley, D.; and Tinos, R. 2016. Efficient hill climber for constrained pseudo-Boolean optimization problems. In *GECCO*, 309–316.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comp.*, 6(2): 182–197.

Derbel, B.; Liefooghe, A.; Zhang, Q.; Aguirre, H.; and Tanaka, K. 2016. Multi-objective local search based on decomposition. In *PPSN*, 431–441.

Dinot, M.; Doerr, B.; Hennebelle, U.; and S., W. 2023. Runtime analyses of multi-objective evolutionary algorithms in the presence of noise. In *IJCAI*, 5549–5557.

Doerr, B.; and Zheng, W. 2021. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *AAAI*, 12293–12301.

Drugan, M. M.; and Thierens, D. 2012. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18: 727–766.

Dubois-Lacoste, J.; López-Ibáñez, M.; and Stützle, T. 2015. Anytime Pareto local search. *Eur. J. Oper. Res.*, 243(2): 369–385.

Figueira, J. R.; Fonseca, C. M.; Halffmann, P.; Klamroth, K.; Paquete, L.; Ruzika, S.; Schulze, B.; Stiglmayr, M.; and Willems, D. 2017. Easy to say they are hard, but hard to see they are easy—towards a categorization of tractable multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 24(1-2): 82–98.

Gao, M.; Liu, C.; and Chen, X. 2025. A bi-objective unrelated parallel machine scheduling problem with additional resources and soft precedence constraints. *Eur. J. Oper. Res.*, 325(1): 53–66.

Garvardt, J.; Grüttemeier, N.; Komusiewicz, C.; and Morawietz, N. 2024. Parameterized local search for max c-Cut. In *IJCAI*, 5586–5594.

Giel, O. 2003. Expected runtimes of a simple multi-objective evolutionary algorithm. In *CEC*, 1918–1925.

Giel, O.; and Lehre, P. K. 2006. On the effect of populations in evolutionary multi-objective optimization. In *GECCO*, 651–658.

Grüttemeier, N.; Komusiewicz, C.; and Morawietz, N. 2021. Efficient Bayesian network structure learning via parameterized local search on topological orderings. In *AAAI*, 12328–12335.

Guo, Z.; Xiao, M.; Zhou, Y.; Zhang, D.; and Tan, K.-L. 2021. Enhancing balanced graph edge partition with effective local search. In *AAAI*, 12336–12343.

Ide, F. H.; Aguirre, H.; and Tanaka, K. 2024. Multi-objective random bit climbers with weighted permutation on large scale binary MNK-landscapes. In *PPSN*, 169–185.

Inja, M.; Kooijman, C.; de Waard, M.; Roijers, D. M.; and Whiteson, S. 2014. Queued Pareto local search for multi-objective optimization. In *PPSN*, 589–599.

Ishibuchi, H.; and Murata, T. 1998. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Trans. Syst., Man, Cybern.*, 28(3): 392–403.

Jaszkiewicz, A. 2002. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment. *IEEE Trans. Evol. Comp.*, 6(4): 402–412.

Jaszkiewicz, A. 2018. Many-objective Pareto local search. *Eur. J. Oper. Res.*, 271(3): 1001–1013.

Johnson, N. L.; Kotz, S.; and Kemp, A. W. 2005. *Univariate Discrete Distributions*. John Wiley & Sons, 3rd edition.

Kang, Y.; Shi, J.; Sun, J.; Zhang, Q.; and Fan, Y. 2024. New techniques to improve neighborhood exploration in Pareto local search. *Expert Syst. Appl.*, 254: 124296.

Knowles, J.; and Corne, D. 2003. Instance generators and test suites for the multiobjective quadratic assignment problem. In *EMO*, 295–310.

Knuth, D. E. 1997. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.

Laumanns, M.; Thiele, L.; Deb, K.; and Zitzler, E. 2002. Combining convergence and diversity in evolutionary multiobjective optimization. *Evol. Comput.*, 10(3): 263–282.

Laumanns, M.; Thiele, L.; and Zitzler, E. 2004. Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. *IEEE Trans. Comp.*, 8(2): 170–182.

Li, M.; Han, X.; Chu, X.; and Liang, Z. 2024. Empirical comparison between MOEAs and local search on multiobjective combinatorial optimisation problems. In *GECCO*, 547–556.

Li, M.; and Yao, X. 2019. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys*, (2).

Liang, J.; Zhou, J.; and Yin, M. 2025. DiverSAT: A novel and effective local search algorithm for diverse SAT problem. In *AAAI*, 11290–11298.

Liang, Z.; and Li, M. 2025. On the problem characteristics of multi-objective pseudo-Boolean functions in runtime analysis. In *FOGA*, 166–177.

Liefooghe, A.; Humeau, J.; Mesmoudi, S.; Jourdan, L.; and Talbi, E.-G. 2012. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18: 317–352.

Lin, P.; Zou, M.; Chen, Z.; and Cai, S. 2024. ParaILP: A parallel local search framework for Integer linear programming with cooperative evolution mechanism. In *IJCAI*, 6949–6957.

Lust, T.; and Teghem, J. 2010. Two-phase Pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3): 475–510.

Lust, T.; and Teghem, J. 2012. The multiobjective multidimensional knapsack problem: a survey and a new approach. *Int. Trans. Oper. Res.*, 19(4): 495–520.

Paquete, L.; Chiarandini, M.; and Stützle, T. 2004. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, 177–199.

Paquete, L.; and Stützle, T. 2003. A two-phase local search for the biobjective traveling salesman problem. In *EMO*, 479–493.

Phan, Q. M.; and Luong, N. H. 2023. Pareto local search is competitive with evolutionary algorithms for multi-Objective neural architecture search. In *GECCO*, 348–356.

Ren, Z.; Srinivasan, A. K.; Vundurthy, B.; Abraham, I.; and Choset, H. 2023. A Pareto-optimal local optimization framework for multiobjective ergodic search. *IEEE Trans. Robot.*, 39(5): 3452–3463.

Ribeiro, C. C.; Hansen, P.; Borges, P. C.; and Hansen, M. P. 2002. A study of global convexity for a multiple objective travelling salesman problem. *Essays and Surveys in Metaheuristics*, 129–150.

Santos, V. L. A.; Carvalho, T. F. M.; de Assis, L. P.; Weiss-Cohen, M.; and Guimarães, F. G. 2022. Multi-objective iterated local search based on decomposition for job scheduling problems with machine deterioration effect. *Engineering Applications of Artificial Intelligence*, 112: 104826.

Shatabda, S.; Newton, M.; and Sattar, A. 2013. Mixed heuristic local search for protein structure prediction. In *AAAI*, 876–882.

Shi, J.; Sun, J.; Zhang, Q.; Zhang, H.; and Fan, Y. 2022. Improving Pareto local search using cooperative parallelism strategies for multiobjective combinatorial optimization. *IEEE Trans. Cybern.*, 2369–2382.

Shi, J.; Zhang, Q.; and Sun, J. 2020. PPLS/D: Parallel Pareto local search based on decomposition. *IEEE Trans. Cybern.*, 50(3): 1060–1071.

Su, Z.; Zhang, Q.; Lü, Z.; Li, C.-M.; Lin, W.; and Ma, F. 2021. Weighting-based variable neighborhood search for optimal camera placement. In *AAAI*, 12400–12408.

Sun, R.; Wang, Y.; Wang, H. S.; Li, H.; Li, X.; and Yin, M. 2024. Nukplex: An efficient local search algorithm for maximum K-plex problem. In *IJCAI*, 7029–7037.

Teghem, J. 1994. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3: 83–104.

Wietheger, S.; and Doerr, B. 2024. Near-tight runtime guarantees for many-objective evolutionary algorithms. In *PPSN*, 153–168.

Xie, Z.; Lü, Z.; Su, Z.; Li, C.-M.; Ding, J.; and Wang, Y. 2024. A swap relaxation-based local search for the latin square completion problem. In *IJCAI*, 7047–7055.

Xie, Z.; Ouyang, Z.; Liu, Q.; and Li, J. 2016. A geometric graph model for citation networks of exponentially growing scientific papers. *Physica A: Statistical Mechanics and its Applications*, 456: 167–175.

Ye, F.; Luo, C.; and Cai, S. 2025. Better understandings and configurations in MaxSAT stochastic local search solvers via anytime performance analysis. In *AAAI*, 27153–27160.

Zhang, Q.; and Li, H. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comp.*, 11(6): 712–731.

Zhang, Z.; Yang, J.; Liu, L.; Xu, X.; Rong, G.; and Feng, Q. 2024. Towards a theoretical understanding of why local search works for clustering with fair-center representation. In *AAAI*, 16953–16960.

Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C.-M.; and Manyà, F. 2025. Integrating multi-armed bandit with local search for MaxSAT. *Artificial Intelligence*, 338: 104242.

Zheng, W.; Li, M.; Deng, R.; and Doerr, B. 2024. How to use the metropolis algorithm for multi-objective optimization? In *AAAI*, 20883–20891.

Zitzler, E.; and Thiele, L. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Trans. Evol. Comp.*, 3(4): 257–271.