Escaping Local Optima with Local Search: A Theory-Driven Discussion

Tobias Friedrich[1], Timo Kötzing[1(✉)], Martin S. Krejca[2], and Amirhossein Rajabi[3]

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{Tobias.Friedrich,Timo.Koetzing}@hpi.de
[2] Sorbonne University, CNRS, LIP6, Paris, France
Martin.Krejca@lip6.fr
[3] Technical University of Denmark, Kgs. Lyngby, Denmark
amraj@dtu.dk

Abstract. Local search is the most basic strategy in optimization settings when no specific problem knowledge is employed. While this strategy finds good solutions for certain optimization problems, it generally suffers from getting stuck in local optima. This stagnation can be avoided if local search is modified. Depending on the optimization landscape, different modifications vary in their success.

We discuss several features of optimization landscapes and give analyses as examples for how they affect the performance of modifications of local search. We consider modifying *random local search* by restarting it and by considering larger search radii. The landscape features we analyze include the *number* of local optima, the *distance* between different optima, as well as the *local landscape* around a local optimum. For each feature, we show which modifications of local search handle them well and which do not.

Keywords: Local search · Theory · Run time analysis

1 Introduction

For optimizing a given objective function, the following strategy is widely used. Start with any, possibly randomly generated, solution. Check *neighboring* solutions, where just a few defining properties of the solution are altered, for having better quality. Whenever you find a better solution, let it replace the previous solution and continue from there. This is the general concept of *local search*.

Basic local search already finds good solutions for a variety of problems [1,15,16,25] by *hillclimbing*, i.e., going up the gradient until a peak in objective value is found. This simple greedy behavior can be very beneficial, e.g., in settings where no additional knowledge about the problem to be optimized is available, so-called *black box optimization*. The main drawback is when local search gets stuck in a local optimum where all nearby solutions do not have better quality

G. Rudolph et al. (Eds.): PPSN 2022, LNCS 13399, pp. 442–455, 2022.
https://doi.org/10.1007/978-3-031-14721-0_31

than the local optimum, while the quality of solutions in other parts of the search space is significantly better. Overcoming the issue of local optima is a long-standing and frequently addressed problem.

One common way to escape local optima is to introduce randomness into how many local changes are performed when modifying a single solution. Prominent examples of this strategy are evolutionary algorithms (EAs [24]), which typically allow to modify solutions to vast extents, larger modifications commonly having a lower probability of occurring. Although this approach potentially allows to escape local optima, it also has some drawbacks. As an example, if better solutions require larger modifications to the current solution, the probability of making such a change may be very small [5]. Moreover, defining a mechanism that allows to change solutions in a manner such that each solution can be produced (a *global* operator) requires greater knowledge of the search space, e.g., when defining the probabilities for each possible change. In contrast, local changes are usually well understood and easy to implement.

In this article, we study *random local search* (RLS), a very basic local-search variant that maintains a single solution. In an iterative manner, it modifies this solution only slightly, i.e., locally. If the new solution is at least as good as the current, the current solution is updated to the new one, otherwise not. It is clear that RLS ceases improving the maintained solution once a search point is found whose direct neighbors have strictly worse objective-function value.

In order to overcome local optima, we consider two simple, different modifications to RLS: *restarts* and *larger search radii*. Restarts modify the way that the maintained solution is selected by always accepting the new solution when a restart is triggered. In addition, the distribution from which the new solution is drawn may be changed. Larger search radii modify the way that a new solution is created by considering solutions that are not direct neighbors of the current solution. This can be done by considering a local operator (i.e., creating solutions in a certain distance) or a global operator (i.e., creating any solution).

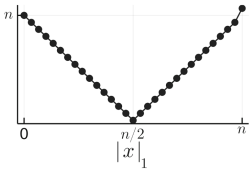
We study RLS and its modifications on various functions (see Fig. 1), containing different types of local optima. Our goal is to understand how the modifications of RLS cope with these local optima. We are particularly interested in an overview of which different characteristics of the optimization landscape favor which modifications and which not.

We aim to raise awareness about the usefulness of modifications to RLS in various settings. To this end, our analyzes do not aim for *depth* (i.e., giving a narrow but sophisticated analysis of a single setting), as is frequent in theory research, but instead for *breadth*. We note that we consider a local optimum to be points in the search space such that all directly neighboring points are worse in objective-function value. Allowing for neighboring points to have equal values results in *plateaus* and in completely different discussions. For recent results on plateaus, we refer the interested reader to the literature [2,4].

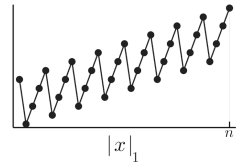
Contributions. Our results concern four landscape characteristics. We give an intuitive description as well as key insights for each characteristic below.

(1) Section 3: A **basin of attraction** of a local optimum x is the part of the landscape from where local search can find the local optimum x.

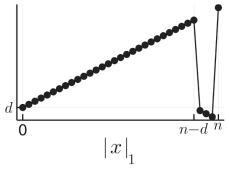
Key Insight: Restarts are beneficial and better than larger search radii if the basin of attraction of the global optimum is large.



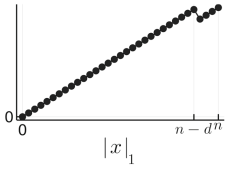
(a) The function TwoMax from Section 3.



(b) The function f_4 as a special case of f_d, defined in Section 6.

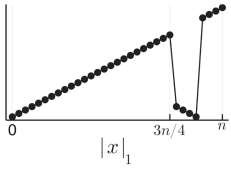


JUMP$_d$

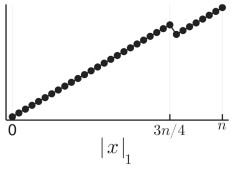


CLIFF$_d$

(c) The two fitness functions defined in Section 4.



SHIFTEDJUMP$_d$



SHIFTEDCLIFF$_d$

(d) JUMP and CLIFF with shifted valleys (Section 5).

Fig. 1. Most of the fitness functions that we analyze in Sects. 3 to 6.

(2) Section 4: Between a local optimum and the global optimum is a *valley* of worse objective-function values that needs to be crossed. Depending on the values within the valley, this is a **deceptive valley** (leading back to the local optimum) or it provides **guiding information**.

Key Insight: Restarts are very beneficial for exploiting guiding information. However, they fail in the case of deception, where larger search radii prove useful and comparable to global operators.

(3) Section 5: The difficulty in crossing valleys depends on whether on the other side of the valley there is a **single or multiple targets** to transition to.

Key Insight: Both modifications of RLS are unaffected by the number of targets. In contrast, a global operator improves the performance drastically.

(4) Section 6: An algorithm might encounter **iterated local optima**, i.e., it has to cross multiple, consecutive valleys to find the global optimum.

Key Insight: The structure of each local optimum is essential. Warm restarts may help majorly if the local structure has guiding information (i.e., is well suited) but fail in case of deceptive information. When using larger search radii, the performance is unaffected by the shape of the valley. It is far slower in case of guiding information but better in case of deception.

Paper Outline. In Sect. 2, we give the details of all algorithms considered, followed by the technical sections considering the four mentioned landscape characteristics in turn. Last, we provide a discussion and conclusions in Sect. 7, where we go into more detail about the general learnings from the analyzes.

Algorithm 1: The framework for trajectory-based heuristics, requiring the potentially parametrized subroutines MUTATE and SELECT as well as a fitness function f.

```
1 x^(0) ← individual drawn uniformly at random from {0,1}^n;
2 for t ∈ ℕ do
3 |   y ← MUTATE(x^(t));
4 |   x^(t+1) ← SELECT_f(x^(t), y);
```

2 Definitions and Algorithms

We let $\mathbb{N}$ denote the set of all natural numbers, including 0, and let $\mathbb{R}$ denote the set of all reals. For all $a, b \in \mathbb{R}$, let $[a..b] := [a, b] \cap \mathbb{N}$ denote the set of natural numbers from at least a to at most b. Further, for all $a \in \mathbb{R}$, let $[a] := [1..a]$.

We consider the maximization of pseudo-Boolean functions of dimension $n \in \mathbb{N}_{\geq 1}$, that is, functions $\{0,1\}^n \to \mathbb{R}$. Throughout this article, let n always denote the dimension of the objective function under consideration. All asymptotics (that is, big-Oh notation) are with respect to this n.

We call a pseudo-Boolean function f a *fitness function*, and we refer to bit strings as *individuals*. For each $x \in \{0,1\}^n$, let $|x|_1$ denote the number of 1s in x, and let $|x|_0$ denote its number of 0s. Further, for each $i \in [n]$, let x_i denote the bit at position i in x. We say that we flip bit i when we refer to the value $1 - x_i$. We call $f(x)$ the *fitness* of x. For $x, y \in \{0,1\}^n$, we call $d_{\mathrm{H}}(x, y) := |\{i \in [n] \mid x_i \neq y_i\}|$ the *Hamming distance* of x and y. Last, for all $i \in [n]$, we call the set of all individuals with distance i to x the *i-neighborhood of x*.

Given an algorithm A and a fitness function f, we call the number of fitness function evaluations (number of calls to f) that A performs until is finds a global maximum of f for the first time the *run time* of A.

2.1 Algorithms

We consider modifications to RLS. All of these algorithms follow the framework of a trajectory-based heuristic for optimizing a fitness function f (Algorithm 1). Each such heuristic evolves iteratively a trajectory $(x^{(t)})_{t \in \mathbb{N}}$ of individuals (the *current* individuals). The initial individual $(x^{(0)})$ is drawn uniformly at random from the search space $\{0,1\}^n$. For all $t \in \mathbb{N}$, the individual $x^{(t+1)}$ is determined via two, potentially parametrized, subroutines: MUTATE and SELECT. The subroutine MUTATE: $\{0,1\}^n \to \{0,1\}^n$ gets $x^{(t)}$ as input (the *parent*) and returns a modified copy of $x^{(t)}$, denoted by y (the *offspring*). We call this process *mutation*, and we say that $x^{(t)}$ is *mutated*. After mutation, utilizing f, the subroutine

SELECT: $(\{0,1\}^n)^2 \to \{0,1\}^n$ selects either $x^{(t)}$ or y as a starting point for the next iteration, and the result is assigned to $x^{(t+1)}$. We refer to this process as *selection*. We allow mutation and selection to take into account additional information, such as the number of iterations since the last improvement was found.

RLS employs *elitist* selection, i.e., if the fitness of the offspring is at least that of the parent, the offspring is selected. During mutation, RLS flips exactly one bit in its parent, which it chooses uniformly at random. Since this approach leads RLS to getting stuck in local optima where the 1-neighborhood is strictly worse, we consider the following modifications of RLS, each of which adjusts selection and/or mutation: restarts and larger search radii.

Restarts. This approach refers to changing selection after a certain amount of non-improving iterations such that it *always* accepts the offspring. In addition, a restart strategy may change how the offspring is generated (i.e., mutation). There are two straightforward ways that we consider: (1) create an individual sampled uniformly at random, that is, start a new run of RLS (*cold* restart), or (2) create offspring normally but always accept it (*warm* restart). We refer to RLS with cold restarts as cr-RLS and to the variant with warm restart as wr-RLS. Both variants have a parameter $R \in \mathbb{R}_{>0}$. If there are more than $n \ln R$ non-improving iterations, the restart is initiated. The parameter R bounds the probability of failing to find the possible improvement. The probability of not finding an improvement in such a situation is at most $1/R$ [22, Lemma 2].

Larger Search Radii. This approach refers to employing mutations that search beyond the 1-neighborhood. One modification following this pattern is *variable neighborhood search* (VNS [10]), for which many different versions exist. We consider the one displayed in Algorithm 2, which creates offspring with increasing distance from the current individual, exploiting each neighborhood fully before going to the next. Each neighborhood is explored randomly, stopping at the first improvement, and each individual in the neighborhood is created at most once. This guarantees to explore all neighborhoods *eventually*. However, as the neighborhood sizes grow exponentially until distance $n/2$ to the parent, it takes a considerable amount of time to get to larger distances.

Adding Global Mutations. Last, we further add a global mutation to RLS in order to see how much the previous algorithms are hampered by relying on local mutations. Since global mutation serves the same purpose as VNS, we remove the VNS modification. The resulting algorithm is effectively an evolutionary algorithm that uses a local search as mutation. This algorithm is called the $(1+1)$ *memetic algorithm* ($(1+1)$ MA [17]; Algorithm 4). After creating its offspring by flipping each of the n bits independently with probability $1/n$, it then aims at improving it via the *first-improvement local search* (FILS; Algorithm 3). FILS creates a random permutation π over $[n]$ and flips each bit of its input in the order they appear in π, keeping those and only those flips that improve the individual. Note that FILS flips bits in potentially improved individuals.

Algorithm 2: VNS maximizing fitness function f.

$x^{(0)} \leftarrow$ individual drawn uniformly at random from $\{0,1\}^n$;
$s \leftarrow 1$;
for $t \in \mathbb{N}$ **do**
  $y \leftarrow x^{(t)}$;
  $\Gamma \leftarrow$ the ordered s-neighborhood of $x^{(t)}$, where the order is chosen uniformly at random;
  for $i \in [|\Gamma|]$ **do**
    $y \leftarrow \Gamma(i)$;
    if $f(y) > f(x^{(t)})$ **then** break the loop iterating over i;
  if $f(y) > f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
    $s \leftarrow 1$;
  else
    $x^{(t+1)} \leftarrow x^{(t)}$;
    $s \leftarrow \min\{s+1, n\}$;

Algorithm 3: First-Improvement Local Search (FILS) of an individual x, maximizing fitness function f.

$\pi \leftarrow$ permutation over $[n]$ chosen uniformly at random;
for $i \in [n]$ **do**
  $y \leftarrow$ copy of x with bit $\pi(i)$ flipped;
  if $f(y) > f(x)$ **then** $x \leftarrow y$;
return x;

3 Basins of Attraction

A basin of attraction [11] is, intuitively, the area of the search space around a local optimum x such that a local-search algorithm ends up in x (in this sense, the local optimum "attracts" the search points in the basin). Note that some search points might lead to different local optima depending on the random choices of the local-search algorithm (in which case they would be counted to all reachable local optima with the probability to reach the local optimum).

A large basin of attraction around a global optimum x^* is good, as it makes it more likely for the local search to find x^*. For the same reason, a large basin of attraction around a local but not global optimum y is bad, as the local search cannot escape y once it gets to its basin of attraction. Thus, the amount and shape of basins of attraction drastically influence how well local search performs.

We briefly discuss this property of search spaces by considering the case of only two local maxima – one being the global maximum. We model this problem via the function TwoMax: $\{0,1\}^n \to \mathbb{R}$ defined in [9], where one local maximum is the all-0s string 0^n, and the other one is the all-1s string 1^n, which is also the

Algorithm 4: (1 + 1) MA maximizing fitness function f.

$x^{(0)} \leftarrow$ individual drawn uniformly at random from $\{0,1\}^n$;
for $t \in \mathbb{N}$ **do**
  $y \leftarrow$ flip each bit in a copy of $x^{(t)}$ with probability $\frac{1}{n}$;
  $z \leftarrow$ apply FILS to y;
  if $f(z) \geq f(x^{(t)})$ **then** $x^{(t+1)} \leftarrow z$;
  else $x^{(t+1)} \leftarrow x^{(t)}$;

Table 1. Results for run times on TwoMax for different algorithms. Highlights show good run times.

Algorithm	Run time	
RLS	∞ with prob. at least 0.5	[14]
cr-RLS, $R = \omega(n)$	**Expected $O(n \log(nR))$**	
wr-RLS, $R = \omega(n)$	$n^{\Omega(n)}$ with prob. at least $(1 - o(1))0.5$	
VNS	$\Omega(2^n)$ with prob. at least 0.5	
(1 + 1) MA	$n^{\Omega(n)}$ with prob. at least 0.5	

unique global maximum.[1] Both maxima have a basin of attraction that consists of an easy slope toward it, and both basins have the same size. More formally, for all $x \in \{0,1\}^n$ we define

$$\textsc{TwoMax}(x) = \begin{cases} n+1, & x = 1^n; \\ \max\{|x|_0, |x|_1\}, & \text{otherwise;} \end{cases}$$

which we aim to maximize; see Fig. 1a for a depiction. Note that a slightly different version of TwoMax, containing both 0^n and 1^n as global maxima, was already defined and analyzed in [18,26].

For this setting we get the following theorem about the performance of various local search algorithms.

Theorem 1. *Regarding run times of RLS, cr-RLS, wr-RLS, VNS, and (1 + 1) MA on* TwoMax*, we get Table 1.*

Intuitively, since the basin of 0^n in TwoMax consists of half the search space, RLS gets stuck at a non-global maximum with probability 1/2 (by symmetry);

[1] The optimum of all test functions in this paper is given by the all-1 string, which leads to the observation that the optimum can be found in constant time by just conjecturing this string. Still theoretical research analyzes such functions, because (a) we can nonetheless observe the behavior of different algorithms on these functions, giving insights into the algorithms; and (b) these functions are representatives of much wider classes of functions with either isomorphic or at least similar properties, but for a theoretical analysis we restrict ourselves to the clean case where the rule "more 1s means closer to the optimum" holds.

this was noted by [14]. For wr-RLS, VNS and the $(1+1)$ MA, the same reasoning applies, with the potential to leave again once stuck at the non-global optimum, but at a stiff price.

Since both basins of TwoMax are large, a cheap way of escaping 0^n is to restart RLS. Choosing a reasonable restart parameter R, the expected run time is not only finite but also very efficient.

Table 2. Results for run times on $\textsc{Jump}_d$ and $\textsc{Cliff}_d$, where $d = O(1)$, $d \geq 2$, for different algorithms. Highlights show good run times.

Algorithm	Run time on $\textsc{Jump}_d$	Run time on $\textsc{Cliff}_d$
RLS	∞ with prob. $1 - o(1)$	∞ with prob. $1 - o(1)$
cr-RLS	Expected $\Omega(2^n)$	Expected $\Omega(2^n/n^d)$
wr-RLS, $R = \Omega(n)$	$n^{\omega(n)}$	**Expected $\boldsymbol{\Theta(n^3 \log R)}$**
VNS	**Expected $\boldsymbol{\Theta(n^d)}$**	Expected $\Theta(n^d)$
$(1+1)$ MA	Expected $\Theta(n^{d+1})$	**Expected $\boldsymbol{\Theta(n^3)}$**

Note that the constant 0.5 is essentially due to the basin of the non-global optimum being a 0.5 portion of the search space. The observation about RLS getting stuck and cr-RLS being efficient can thus be generalized in dependence of how large the basin of the global optimum is. We omit this generalization.

4 Deceptive Valleys vs. Guiding Information

Given a local optimum, a *valley* is the area of the search space that has lower fitness than the local optimum but that has to be crossed to arrive at the global optimum. We consider crossing two kinds of valleys. Two well-established fitness functions to model this setting are Jump [8] and Cliff [12], parametrized by $d \in \mathbb{N}$, determining the width of the valley. The two functions model two extremes regarding the shape of the valley: In Jump, the valley contains deceptive fitness signals, guiding the search back to the local optimum, while in Cliff the fitness signal points to the global optimum. Formally, for all $x \in \{0,1\}^n$, let

$$\textsc{Jump}_d(x) = \begin{cases} |x|_1 + d, & \text{if } |x|_1 \leq n - d \vee |x|_1 = n; \\ |x|_0, & \text{otherwise;} \end{cases}$$

$$\textsc{Cliff}_d(x) = \begin{cases} |x|_1, & \text{if } |x|_1 \leq n - d; \\ |x|_1 - d + 1/2, & \text{otherwise.} \end{cases}$$

Both functions are functions of unitation, i.e., the fitness only depends on the number of 1s of the evaluated solution (see Fig. 1c). Note that there are far more search points with about $n/2$ 0s than with just a few 0s (where the valley is), so any local search starts, with high probability, somewhere in the middle and

encounters the valley on the way to the global optimum. As a result, with high probability, RLS ends up in a local optimum without chance of escaping. Thus, cold restarts do not lead to successful optimization in polynomial time.

One way to overcome the valley is by finding a local optimum (in distance d of the global optimum) and then creating the global optimum with a single mutation. This is what VNS does. Note that, in this case, the exact layout of the valley is of no importance. This is very different for algorithms which can explore valleys. The (1 + 1) MA and wr-RLS both suffer from the presence of deceptive information, while making good use of guiding information.

Theorem 2. *Regarding run times of RLS, cr-RLS, wr-RLS, VNS, and (1 + 1) MA on* Jump *and* Cliff*, we get Table 2.*

The idea of the proof for the (1 + 1) MA is as follows. When currently in a local optimum, with probability $\Theta(1/n)$, samples a search point in the valley just one step closer to the optimum and then, with probability $\Theta(1/n)$ runs up the slope to the global optimum (an otherwise returns to the local optimum).

5 Single Target vs. Multiple Targets

In Sect. 4, we discuss crossing a valley to reach one specific point. In this section, we address the question of what changes if there is not just one point on the other side of the valley, but multiple. To this end, we consider again two fitness functions; they are variants of Jump and Cliff from Sect. 4 but suitably shifted into an area of the search space with more than one point after the valley. The case of Jump was first considered in [3,21]. We make the following formal definitions. Let $d \in \mathbb{N}$. For all $x \in \{0,1\}^n$,

$$\textsc{ShiftedJump}_d(x) = \begin{cases} |x|_1 + d, & \text{if } |x|_1 \leq 3n/4 \text{ or } |x|_1 \geq 3n/4 + d; \\ |x|_0, & \text{otherwise;} \end{cases}$$

$$\textsc{ShiftedCliff}_d(x) = \begin{cases} |x|_1, & \text{if } |x|_1 \leq 3n/4; \\ |x|_1 - d + 1/2, & \text{otherwise.} \end{cases}$$

Note that the depictions of the functions in Fig. 1d are somewhat misleading: It looks like there is still only one solution directly after the valley. However, since the search space is not the integers from 0 to n, but rather all bit strings $\{0,1\}^n$, there are indeed a lot of points on the other side of the valley at a distance of d to any local optimum: A local optimum has exactly $n/4$ many 0s, and flipping any d of those 0s gives a solution on the other side of the valley (i.e., a point with a fitness higher than that of the local optimum). Thus, for constant d, there are indeed $\Theta(n^d)$ search points just on the other side of the valley.

In Sect. 4, we show that the VNS and the (1 + 1) MA behave basically the same for crossing a deceptive valley: they need to make the jump to the other side of the valley in one go. In this section, we show a major difference. For VNS, after finding a local optimum, this algorithm first searches neighborhoods

of distance less than d before finally picking a distance of d for the search. This implies that a lot of time is wasted searching through unrewarding parts of the search space. In contrast to this, the global mutation of the (1 + 1) MA enables stepping over the valley in one jump *of constant probability*. This is also the behavior exhibited by the (1 + 1) EA (see [3]).

Theorem 3. *Regarding run times of RLS, cr-RLS, wr-RLS, VNS, and (1 + 1) MA on* ShiftedJump *and* ShiftedCliff*, we get Table 3.*

Table 3. Results for run times on $\textsc{ShiftedJump}_d$ and $\textsc{ShiftedCliff}_d$, where $d = O(1)$, $d \geq 2$, for different algorithms. Highlights show good run times.

Algorithm	$\textsc{ShiftedJump}_d$	$\textsc{ShiftedCliff}_d$
RLS	∞ with prob. $1 - o(1)$	∞ with prob. $1 - o(1)$
cr-RLS	Expected $2^{\Omega(n)}$	Expected $\Omega(2^n \binom{n}{n/4-1}^{-1})$
wr-RLS, $R = \Omega(n)$	$n^{\omega(n)}$	$\boldsymbol{\Theta(n \log(R))}$
VNS	Expected $\Theta(n^{d-1})$	Expected $\Theta(n^{d-1})$
(1 + 1) MA	**Expected $\boldsymbol{\Theta(n)}$**	**Expected $\boldsymbol{\Theta(n)}$**

6 Iterated Local Optima

In Sect. 4, we show that non-elitist algorithms can have a big advantage in crossing fitness valleys. In this section, we point out one drawback of such algorithms, namely that they can fail and essentially have to restart optimization from a bad part of the search space. Let us suppose, e.g., that the valley is crossed successfully with probability p and otherwise a complete reoptimization has to be made. If only a single valley has to be crossed, this success probability gives $1/p$ attempts and reoptimizations in expectation, which might still be acceptable. However, the success probability decreases exponentially with the number of optima to be crossed in approaching the global optimum.

This is modeled by the following fitness functions inspired by combining the Hurdle fitness function [19] and the Ridge fitness function [20]. Hurdle consists of multiple Cliff-like structures, leading to a sequence of local optima. Ridge is a fitness function where the algorithm has a path of "width 1" to climb to go up to the global optimum. In order to make comparisons with only one local optimum on a ridge, we also define a version of Cliff on a ridge. For any

$d \in \mathbb{N}$ (denoting the length of the valley) and for all $i \in [0..n]$ and $x \in \{0,1\}^n$,

$$f_d(i) = \begin{cases} 2n, & \text{if } i = n; \\ f_d(i+1) + 2d - 3, & \text{if } d \text{ divides } n - i; \\ f_d(i+1) - 2, & \text{otherwise;} \end{cases}$$

$$\textsc{HurdleRidge}_d(x) = \begin{cases} n + f_d(|x|_1), & \text{if } x = 1^{|x|_1}0^{n-|x|_1}; \\ |x|_0, & \text{otherwise.} \end{cases}$$

$$\textsc{CliffRidge}_d(x) = \begin{cases} |x|_0, & \text{if } x \neq 1^{|x|_1}0^{n-|x|_1}; \\ n + |x|_1, & \text{if } |x|_1 \leq n - d; \\ n + |x|_1 - d + 1/2, & \text{otherwise.} \end{cases}$$

Note that, for $i \in [0..n]$, (see also Fig. 1b for a depiction)

$$f_d(i) = 2i - (2d+1)|\{j \in [i..n-1] \mid d \text{ divides } n - i\}|.$$

Most search points in HurdleRidge point to the solution 0^n; this is the starting point of the path to the global optimum 1^n. Along this path the fitness is steadily increasing, but once every d steps it goes down $d - 1$, leading to $\Theta(n/d)$ valleys of width d to be crossed.

Table 4. Results for run times on $\textsc{HurdleRidge}_d$ and $\textsc{CliffRidge}_d$, where $d = O(1)$, $d \geq 2$, for different algorithms. Highlights show good run times.

Algorithm	$\textsc{HurdleRidge}_d$	$\textsc{CliffRidge}_d$
RLS	∞ with prob. $1 - o(1)$	∞ with prob. $1 - o(1)$
cr-RLS	Expected $2^{\Omega(n)}$	Expected $2^{\Omega(n)}$
wr-RLS, $R = \Omega(n \log n)$	$\boldsymbol{O(n^3 + n^2 \log R)}$	$\forall c : \Omega(n^c)$ with prob. $1 - o(1)$
VNS	Expected $\Theta(n^{d+1})$	Expected $\Theta(n^d)$
(1 + 1) MA	**Expected $\boldsymbol{\Theta(n^3)}$**	**Expected $\boldsymbol{\Theta(n^3)}$**

For elitist algorithms, optimization proceeds by crossing each of the $\Theta(n)$ many local optima one after the other. In contrast to this result, non-elitist algorithms have a chance to *fail* crossing a fitness valley. This is not a big problem if there is only one valley to be crossed, resulting in an acceptable optimization time on CliffRidge. But on HurdleRidge there are linearly many valleys to cross, so even some small failure probability per crossing leads almost surely to failing to optimize. This happens for warm restarts for $\textsc{HurdleRidge}_d$. Note that this result does not generalize to the (1 + 1) MA, since it can recover from a failure when trying to cross a valley by reverting to the best-so-far solution.

Theorem 4. *Regarding run times of RLS, cr-RLS, wr-RLS, VNS and (1 + 1) MA on* HurdleRidge *and* CliffRidge*, we get Table 4.*

7 Discussion and Conclusion

We have seen many different strategies for overcoming local optima. The first strategy, applicable to any randomized algorithm, is to just run the algorithm multiple times (cr-RLS). This leads to a very diverse set of starting points for local search and can boost the success probability of any algorithm which starts off with a reasonable success probability. One problem in this area is to decide when to restart. For RLS, this decision is somewhat easily made, since after about $n \log n$ iterations without improvements, all neighbors have been considered at least once with high probability, so no further improvement occurs. In practice, also small improvements might be a sign of stagnation and can be used as a signal to restart the algorithm. An extreme version of searching with restarts is random search, where no local optimization is employed. This strategy is popular when the fitness landscape is extremely rugged (which blocks local optimization) and different parts of the landscape are very different. Simple grid search optimization also falls into this category.

In Sect. 4, we have seen that giving up elitism in favor of being able to make use of guiding information in the valley might be valuable. Some of the first algorithms that made use of this idea were the Metropolis algorithm and Simulated Annealing, which in turn suffer in their ability to climb simple gradients; for a theoretical comparison with elitist search heuristics, see [13]. Both the Metropolis Algorithm and Simulated Annealing behave like RLS, but they accept worse offspring with a certain probability depending on the fitness difference to the parent. This makes the algorithms sensitive to the fitness values, in contrast to the non-elitist (and elitist) algorithms considered in this paper based on restarts (accepting worse moves only rarely). The advantage of rare (warm) restarts is that other moves can be elitist and thus able to find local optima. Since there are typically more potential worsening moves than improving moves, it is vital to reject worsening moves most of the time.

Another strategy for overcoming local optima is to look further than just the direct neighborhood. This is the idea behind VNS. However, sometimes a lot of samples are wasted locally before attempting a larger jump as, for example, $(1 + 1)$ MA does, see Sect. 5. This is the principle domain of global search heuristics, such as the well-studied $(1 + 1)$ EA. Taking this idea one step further gives the so-called fast $(1 + 1)$ EA [6], sampling offspring at far distances significantly more frequently than the $(1 + 1)$ EA, while still sampling search points at a distance of 1 with constant probability. Another idea is to adjust the search distance distribution whenever progress stagnates; this idea, so-called *stagnation detection*, was analyzed in [7,21–23]. Note that it is typically fruitful to spend a lot of time searching the local neighborhood in order to exploit local structure.

The different test functions considered are abstractions of what real-world optimization problems look like. In particular, they study the different features in isolation. In Sect. 6, we discussed a test function where a complex test function is constructed by iterating the setting of a local optimum. We saw that in this more complex setting, an algorithm that is successful without this iterated setting is now unsuccessful. Iterated obstacles are generally no bigger problem for elitist

algorithms than non-iterated obstacles, but non-elitism has to be applied more carefully. The (1 + 1) MA provides a hybrid, where non-elitism is allowed, but the algorithm might revert to the best-so-far search point.

In conclusion, we see that there is no universally best strategy to do so (which is known for a long time), but properties of the fitness landscape can inform about what algorithms could be efficient. In this paper, we studied the connections between the properties of the fitness landscape and the success of various strategies. In general, since most of the variants do not hamper the ability of local search to find local optima, it is advisable to use *some* variant that can escape local optima. However, the choice of *which* variant to choose depends on the fitness landscape of the problem to optimize. Thus, if one has some knowledge about the optimization problem, that is, one faces a *gray*-box and not a *black*-box scenario, incorporating this knowledge into the choice of how to escape local optima is a very useful or even crucial step in order to get best possible results.

Acknowledgments. This work was supported by a grant by the Independent Research Fund Denmark (DFF-FNU 8021-00260B), and by the Paris Île-de-France Region via the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 945298-ParisRegionFP.

References

1. Aarts, E., Aarts, E.H., Lenstra, J.K.: Local Search in Combinatorial Optimization. Princeton University Press, Princeton (2003)
2. Antipov, D., Doerr, B.: Precise runtime analysis for plateau functions. ACM Trans. Evol. Learn. Optim. **1**(4), 13:1–13:28 (2021). https://doi.org/10.1145/3469800
3. Bambury, H., Bultel, A., Doerr, B.: Generalized jump functions. In: Proceedings of GECCO 2021, pp. 1124–1132. ACM (2021). https://doi.org/10.1145/3449639.3459367
4. Bian, C., Qian, C., Tang, K., Yu, Y.: Running time analysis of the (1+1)-EA for robust linear optimization. Theor. Comput. Sci. **843**, 57–72 (2020). https://doi.org/10.1016/j.tcs.2020.07.001
5. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of GECCO 2017, pp. 777–784. ACM Press (2017)
6. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Bosman, P.A.N. (ed.) Proceedings of GECCO 2017, pp. 777–784. ACM (2017). https://doi.org/10.1145/3071178.3071301
7. Doerr, B., Rajabi, A.: Stagnation detection meets fast mutation. In: Proceedings of EvoCOP 2022, pp. 191–207. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-04148-8_13
8. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**, 51–81 (2002)
9. Friedrich, T., Oliveto, P.S., Sudholt, D., Witt, C.: Analysis of diversity-preserving mechanisms for global exploration. Evol. Comput. **17**(4), 455–476 (2009)
10. Hansen, P., Mladenovic, N.: Variable neighborhood search. In: Martí, R., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Heuristics, pp. 759–787. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-07124-4_19

11. Horn, J., Goldberg, D.E.: Genetic algorithm difficulty and the modality of fitness landscapes. In: Proceedings of FOGA 1995, vol. 3, pp. 243–269. Elsevier (1995)
12. Jagerskupper, J., Storch, T.: When the plus strategy outperforms the comma strategy and when not. In: 2007 IEEE Symposium on Foundations of Computational Intelligence, pp. 25–32. IEEE (2007)
13. Jansen, T., Wegener, I.: A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. Theor. Comput. Sci. **386**(1), 73–93 (2007). https://doi.org/10.1016/j.tcs.2007.06.003, https://www.sciencedirect.com/science/article/pii/S0304397507004811
14. Jansen, T., Zarges, C.: Example landscapes to support analysis of multimodal optimisation. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 792–802. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_74
15. Johnson, D.S.: Local optimization and the Traveling Salesman Problem. In: Paterson, M.S. (ed.) ICALP 1990. LNCS, vol. 443, pp. 446–461. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0032050
16. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity. Springer, Cham (2010). https://doi.org/10.1007/978-3-642-16544-3
17. Nguyen, P.T.H., Sudholt, D.: Memetic algorithms outperform evolutionary algorithms in multimodal optimisation. Artif. Intell. **287**, 103345 (2020). https://doi.org/10.1016/j.artint.2020.103345
18. Pelikan, M., Goldberg, D.E.: Genetic algorithms, clustering, and the breaking of symmetry. In: Schoenauer, M., et al. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 385–394. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_38
19. Prügel-Bennett, A.: When a genetic algorithm outperforms hill-climbing. Theoret. Comput. Sci. **320**(1), 135–153 (2004)
20. Quick, R.J., Rayward-Smith, V.J., Smith, G.D.: Fitness distance correlation and Ridge functions. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 77–86. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0056851
21. Rajabi, A., Witt, C.: Stagnation detection in highly multimodal fitness landscapes. In: Proceedings of GECCO 2021. ACM Press (2021)
22. Rajabi, A., Witt, C.: Stagnation detection with randomized local search. In: Zarges, C., Verel, S. (eds.) EvoCOP 2021. LNCS, vol. 12692, pp. 152–168. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72904-2_10
23. Rajabi, A., Witt, C.: Self-adjusting evolutionary algorithms for multimodal optimization. Algorithmica **84**, 1694–1723 (2022). https://doi.org/10.1007/s00453-022-00933-z. Preliminary version in GECCO 2020
24. Simon, D.: Evolutionary Optimization Algorithms. Wiley, Hoboken (2013)
25. Stützle, T.: Applying iterated local search to the permutation flow shop problem. Technical report, Citeseer (1998)
26. Van Hoyweghen, C., Goldberg, D.E., Naudts, B.: From TwoMax to the Ising model: easy and hard symmetrical problems. Generations **11**(01), 10 (2001)