The Permutation in a Haystack Problem and the Calculus of Search Landscapes

Vincent A. Cicirello, *Member, IEEE*

***Abstract*—The natural encoding for many search and optimization problems is the permutation, such as the traveling salesperson, vehicle routing, scheduling, assignment and mapping problems, among others. The effectiveness of a given mutation or crossover operator depends upon the nature of what the permutation represents. For some problems, it is the absolute locations of the elements that most directly influences solution fitness; while for others, element adjacencies or even element precedences are most important. Different permutation operators respect different properties. We aim to provide the genetic algorithm or metaheuristic practitioner with a framework enabling effective permutation search landscape analysis. To this end, we contribute a new family of optimization problems, the *Permutation in a Haystack*, that can be parameterized to the various types of permutation problem (e.g., absolute vs relative positioning). Additionally, we propose a *Calculus of Search Landscapes*, enabling analysis of search landscapes through examination of local fitness rates of change. We use our approach to analyze the behavior of common permutation mutation operators on a variety of Permutation in a Haystack landscapes; and empirically validate the prescriptive power of the search landscape calculus via experiments with simulated annealing.**

***Index Terms*—search landscape analysis, permutation representation, mutation operators, genetic algorithm, simulated annealing, permutation distance metrics**

I. Introduction

THE natural encoding for many search and optimization problems is the permutation, such as any problem where an optimal ordering over the elements of a set is needed (e.g., the traveling salesperson (TSP), vehicle routing, scheduling problems, etc). Assignment and mapping problems, where we must map the elements of one set to the elements of another set, are also naturally encoded as permutations (e.g., fix the order of the elements of one set, and a permutation of the other set represents the mapping). Even a problem like bin-packing can be encoded with a permutation, representing the order items are placed into bins via some rule, such as first-fit.

Permutation problems can be categorized into three broad classes. Campos et al. discuss two of these, *A-Permutation* and *R-Permutation* problems, in depth [1], [2]. For A-Permutation problems, absolute element positions most directly impact permutation fitness (e.g., assignment and mapping problems). For R-Permutation problems, relative ordering (i.e., element adjacency) is most indicative of solution fitness (e.g., problems like the TSP where the permutation represents a set of edges). We add to Campos et al's classification scheme a third category, *P-Permutation* problems, in which the *precedence* among the elements (and not just direct adjacency) impacts fitness, such as if the permutation represents rankings (e.g., problems in information retrieval and recommender systems [3]–[5]). For example, perhaps it is important to fitness that element w ranks higher than elements $\{x, y, z\}$ but that it does not particularly matter which of x, y, or z (or even some other element q) immediately follows w. Fig. 1 summarizes the features with greatest impact on fitness by problem class.

Because permutations have such wide-ranging problem-dependent interpretations, many neighborhood, mutation, and crossover operators have been developed for searching the space of permutations, focusing on different characteristics. For example, cycle crossover retains absolute element positions [6]. Partially Matched Crossover (PMX) [7] and its uniform counterpart UPMX [8], though more disruptive, are also predominantly A-Permutation operators. Others, such as order crossover, retain element adjacencies [9]; or in the case of non-wrapping order crossover, focus on maintaining adjacency while minimizing positional displacement [10]. Later in Section IV we discuss in depth the most common mutation and neighborhood operators for permutation problems.

Although there has been research on permutation search landscapes (e.g., [11]–[16]), often the relevance of permutation operators for problems is discussed in less formal terms; or in some cases, it is less clear how to apply the existing theory of search landscapes. For example, it may be non-trivial to identify an appropriate distance metric for characterizing the fitness landscape (e.g., computing the "natural" distance metric for a Reversal mutation is NP-Hard [13], [17]).

The objective of this paper is to help bridge the gap in the theory of permutation search landscapes. Our first contribution in this direction is the *Permutation in a Haystack* problem, introduced as a new framework for studying permutation search landscapes (Section III). The Haystack problem uses permutation distance metrics to specify search landscape topology, providing an easy means of studying the behavior of search operators on a wide variety of permutation landscapes from the three major categories: A-Permutation, R-Permutation, and P-Permutation. We include an analysis of fitness-distance correlation (FDC) for the most common permutation mutation operators on several Haystack search landscapes (Section V).

The second of our contributions to the theory of permutation search is the development of a *Calculus of Search Landscapes*,

V. A. Cicirello is with the Computer Science and Information Systems Program, School of Business, Stockton University, Galloway, NJ, 08205 USA e-mail: cicirelv@stockton.edu.

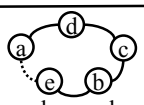
Problem Class	A-Permutation Problems	R-Permutation Problems	P-Permutation Problems
Features with largest impact on fitness	[a d c b e] 0 1 2 3 4 positions only	 edges only	 pairwise precedences

Fig. 1. Permutation features with greatest impact on fitness by problem class.

a new approach to search landscape analysis that focuses on local fitness rates of change (Section VI). We introduce the concept using examples of its application to the more common bit-string encoding and to real-valued function optimization. Thus, though we focus on permutations in this paper, the theory is also relevant to other common encodings. We then apply the search landscape calculus in an in-depth analysis of the Haystack family of search problems for our collection of permutation mutation operators. We empirically validate the approach in Section VII using simulated annealing (SA), demonstrating the power of the search landscape calculus in identifying promising neighborhood operators. We now begin with a review of background and related work.

II. Background and Related Work

We can conceptualize the space of candidate solutions to an optimization problem as points on a surface where the height of the point corresponds to the fitness of that solution. This fitness landscape [18], a concept from natural genetics [19], likely contains a variety of peaks and valleys which represent local optima. The optimization problem is to find an optimal point on that landscape. There is great interest in studying properties of search landscapes that influence optimization hardness. Different properties of the fitness landscape lead to more productive search for some algorithms, and less productive for others. Mitchell et al. developed a family of fitness functions, the *Royal Roads*, enabling the study of how different combinations of fitness landscape features affect the performance of a genetic algorithm (GA)—e.g., features such as isolated regions of high fitness, multiple conflicting solutions, etc [20], [21]. The royal roads, and other similar fitness functions, have been used by many to better learn when GAs perform well (e.g., [22]–[24]). Many others have studied *GA-deceptive* search landscapes (e.g., [25]–[27])—search landscapes where GAs tend to be deceived by high average fitness low-order schema that do not recombine into high average fitness high-order schema [28].

An important characteristic of a search landscape is FDC [29], which is the Pearson correlation coefficient applied to measure the correlation between the fitness of a solution and its distance to the nearest optimal solution. If fitness (i.e., solution quality) improves the nearer you are to an optimal solution (as distance to the optimal solution decreases), then searching locally (i.e., via mutation) around the current population of solutions will be effective (or around the current solution for non-population based metaheuristics such as SA).

Search landscape topology depends on the fitness function as well as the neighborhood operator, which controls which candidate solutions can be reached in one step. For the bit-string representation of a GA, Hamming distance is commonly used. Though this may be appropriate for characterizing the search landscape for some operators (e.g., bit-flip mutation), others have explored alternatives when studying search landscapes induced by crossover. For example, Höhn and Reeves argue that the Hamming landscape does not adequately characterize crossover behavior, and define a family of crossover and complementary crossover landscapes for OneMax [30].

We are specifically interested in the study of problems that involve optimizing permutation-based structures. To analyze search landscapes for permutation problems, we need a means of measuring the distance between permutations. Schiavinotto and Stützle argue that the distance should be the minimum number of applications of the operator needed to transform one candidate solution into the other [13]. In other words, the ideal distance metric for fitness landscape analysis is an "edit distance" with the neighborhood operator as the edit operation. The edit distance between two structures (e.g., permutations) is the minimum cost of the "edit operations" required to transform one structure into the other.[1] The concept originated in the string pattern matching community [31] and is easily extended to permutations [15]. It is often necessary or desirable to instead use a "surrogate" distance, such as when the natural distance for the operator is either too costly to compute [13], or perhaps otherwise escapes definition due to the complexity of the operator. A wide variety of permutation distance measures, including edit distances, can be found in the literature which capture different aspects of a permutation, such as absolute positions [14]–[16], [32], [33], relative positions [1], [2], [34], [35], or element precedences [3], [14], [15], [36], [37].

III. Permutation in a Haystack

We define the *Permutation in a Haystack* problem, Haystack(δ, N), as follows:

Haystack(δ, N)**:** Find the permutation p such that:

$$p = \underset{p' \in S_N}{\operatorname{argmin}}\, \delta(p', p_N), \tag{1}$$

where S_N is the set of all permutations of the set $\{0, 1, \ldots, (N-1)\}$, $p_N = [0, 1, \ldots, (N-1)]$ is a permutation with the N elements in increasing order, and δ is a distance function on permutations which serves as the optimization objective function.

Obviously the optimal solution to this problem is simply $p = p_N$, the "needle" in our figurative "haystack." However, the choice of distance function δ affects the terrain of the landscape and thus search performance. We intend for this problem to be the permutation analog to the well-known *OneMax* problem for bit-strings, where the problem is to minimize hamming distance to the string of all ones.

Throughout the paper, we use the following notation: $p(i)$ refers to the element in position i of permutation p.

We consider several different choices for δ. In deciding which distance measures to include, we surveyed the literature on permutation distance and have included a large variety of

[1] We do not refer specifically to the Levenshtein distance.

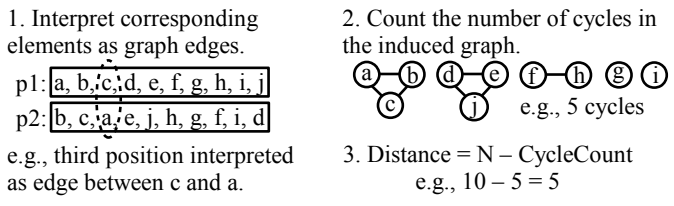


Fig. 2. Example of computing interchange distance by counting cycles.

distance functions to capture the essence of a wide variety of optimization problems from the three major categories: A-Permutation, R-Permutation, and P-Permutation problems.

A. A-Permutation Distance Measures

Exact Match: Introduced by Ronald, the Exact Match (EM) distance, Hamming distance extended to non-binary strings, is the number of permutation positions containing different elements [32]. EM is widely used (e.g., [14]–[16], [33]), and satisfies the metric properties [32]. We define it as:

$$\delta_{\text{EM}}(p_1, p_2) = \sum_{i=1}^{N} \begin{cases} 1 & \text{if } p_1(i) \neq p_2(i) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Interchange Distance: In our prior work, we used an edit distance, interchange distance, with element interchanges or swaps as the edit operation [14], [16]. Like all edit distances, it is a metric. We refer to general swaps, and not to strict adjacent swaps; and define $\delta_{\text{Interchange}}(p_1, p_2)$ as the minimum number of swaps needed to transform p_1 into p_2.

Interchange distance is computed efficiently by counting the number of cycles between the permutations [14]. A permutation cycle of length k is transformed into k fixed points with $k-1$ swaps (a fixed point is a cycle of length 1). Fig. 2 illustrates computing interchange distance by cycle counting. Let CycleCount(p_1, p_2) be the number of permutation cycles. Thus, we can formalize interchange distance as:

$$\delta_{\text{Interchange}}(p_1, p_2) = N - \text{CycleCount}(p_1, p_2). \tag{3}$$

Deviation Distance: The deviation distance, which originated with Ronald, is the sum of the positional deviations of the permutation elements [32]. It is found in two forms in the literature, including Ronald's version which we define with:

$$\delta_{\text{Dev1}}(p_1, p_2) = \tfrac{1}{N-1} \sum_{e \in p_1} |i - j|, \text{ where } p_1(i) = p_2(j) = e. \tag{4}$$

Several use Ronald's version of deviation distance (e.g., [15]) including our own prior work [14], [16]; while others (e.g., [1], [33]) drop the $1/(N-1)$ and instead use:

$$\delta_{\text{Dev2}}(p_1, p_2) = \sum_{e \in p_1} |i - j|, \text{ where } p_1(i) = p_2(j) = e. \tag{5}$$

Ronald divides by $N-1$ to bound an individual element's contribution to the total distance in the interval $[0, 1]$, since an element is displaced by at most $N-1$ (e.g., if it is at one end of one permutation, and the other end of the second). Depending on the application, the version used is of minimal relevance, at best, since scaling to any desired maximum distance is straightforward. Deviation distance is a metric [32].

B. R-Permutation Distance Measures

Edge Distance: Ronald also considers permutation distance measures appropriate when relative ordering of elements is most important rather than absolute positions, and defines both the cyclic edge distance and the acyclic edge distance [34]. Both forms assume that the adjacency of elements corresponds to undirected edges, and that the absolute positions otherwise are unimportant. Cyclic edge distance considers the permutation to be a cyclic structure where the first and last elements are adjacent; whereas acyclic edge distance does not. Cyclic edge distance interprets the permutation, $[a, b, c, d, e]$, as the set of undirected edges, $\{(a, b), (b, c), (c, d), (d, e), (e, a)\}$, while acyclic edge distance interprets it as the set, $\{(a, b), (b, c), (c, d), (d, e)\}$. For both interpretations, permutations are invariant under complete inversions (e.g., $[a, b, c, d, e]$ is equivalent to $[e, d, c, b, a]$). For the cyclic interpretation, permutations are also invariant under rotations. In both forms, the distance between two permutations is the number of non-shared edges. It is relevant in characterizing the fitness landscapes of problems such as the TSP where it is the relative ordering of the permutation elements that directly impacts solution fitness, rather than the absolute locations [35]. Ronald showed that both are semi-metrics (due to inversion invariance, and due to rotational invariance for cyclic edge distance); and suggested a canonical permutation form that induces satisfaction of all of the metric properties [34]. We do not assume that canonical form in our work; and thus the Haystack(δ, N) problem with edge distance has multiple solutions. We formalize the cyclic and acyclic edge distances, respectively, as follows:

$$\delta_{\substack{\text{Cyclic} \\ \text{Edge}}}(p_1, p_2) = \sum_{i=1}^{N} \begin{cases} 0 & \text{if } \exists j \exists x \exists y,\ j = (i \bmod N) + 1 \wedge \\ & \quad y = (x \bmod N) + 1 \wedge \\ & \quad [(p_1(i) = p_2(x) \wedge p_1(j) = p_2(y)) \\ & \quad \vee \\ & \quad (p_1(i) = p_2(y) \wedge p_1(j) = p_2(x))] \\ 1 & \text{otherwise.} \end{cases} \tag{6}$$

$$\delta_{\substack{\text{Acyclic} \\ \text{Edge}}}(p_1, p_2) = \sum_{i=1}^{N-1} \begin{cases} 0 & \text{if } \exists x,\ (p_1(i) = p_2(x) \wedge \\ & \quad p_1(i+1) = p_2(x+1)) \vee \\ & \quad (p_1(i) = p_2(x+1) \wedge \\ & \quad p_1(i+1) = p_2(x)) \\ 1 & \text{otherwise.} \end{cases} \tag{7}$$

R-Type Distance: Another relative ordering distance measure is R-type distance [1], [2]. It is closely related to acyclic edge distance. However, R-type distance interprets adjacent permutation elements as directed edges. We define it as:

$$\delta_{\text{RType}}(p_1, p_2) = \sum_{i=1}^{N-1} \begin{cases} 0 & \text{if } \exists x,\ p_1(i) = p_2(x) \wedge \\ & \quad p_1(i+1) = p_2(x+1) \\ 1 & \text{otherwise.} \end{cases} \tag{8}$$

Cyclic R-type distance, though not previously described in the literature, is a cyclic counterpart to R-type distance that considers there to be an edge from the last element to the first. It is more relevant for problems such as the Asymmetric

TSP, where there is such an edge, but for which cyclic edge distance is irrelevant since the distance between cities depends on direction of travel. We define cyclic R-type distance as:

$$\delta_{\substack{\text{Cyclic}\\\text{RType}}}(p_1,p_2)=\sum_{i=1}^{N}\begin{cases}0 & \text{if } \exists j\exists x\exists y,\ j=(i \bmod N)+1\ \wedge\\ & \quad y=(x \bmod N)+1\ \wedge\\ & \quad (p_1(i)=p_2(x)\wedge p_1(j)=p_2(y))\\ 1 & \text{otherwise.}\end{cases} \tag{9}$$

Though R-type distance satisfies the metric properties, cyclic R-type is only a semi-metric due to rotational invariance.

C. P-Permutation Distance Measures

Kendall Tau Distance: Kendall tau distance, a metric, is relevant when permutations represent rankings [36]. Whereas all forms of edge distance and R-type distance focus on adjacencies; Kendall tau focuses on general relative positions indicative of element precedence. It is defined as follows:

$$\delta_\tau(p_1,p_2)=\sum_{i=1}^{N-1}\sum_{j=(i+1)}^{N}\begin{cases}0 & \text{if } \exists x\exists y,\ p_1(i)=p_2(x)\ \wedge\\ & \quad p_1(j)=p_2(y)\ \wedge\ x<y\\ 1 & \text{otherwise.}\end{cases} \tag{10}$$

Kendall originally normalized by dividing by the maximum, $N(N-1)/2$ [36], but others do not (e.g., [3], [37]).

Reinsertion Distance: Reinsertion distance is an edit distance; and thus satisfies the metric properties. It considers a single atomic edit operation, removal/reinsertion, which removes an element and reinserts it elsewhere in the permutation. Thus, $\delta_{\text{Reinsertion}}(p_1,p_2)$ is the minimum removal/reinsertions needed to transform p_1 into p_2.

In our prior work [14], we adapted a dynamic programming algorithm for string edit distance [31] to compute reinsertion distance by assigning edit operation costs of 0.5 for removals, 0.5 for insertions, and ∞ for replacements (to disallow them).

Another efficient way to compute reinsertion distance relies on the observation: the elements that must be removed and reinserted are exactly the elements that do not lie on the longest non-contiguous subsequence common to both permutations. Thus, we can compute reinsertion distance as:

$$\delta_{\text{Reinsert}}(p_1,p_2)=N-\#(\text{MaxCommonSubsequence}(p_1,p_2)). \tag{11}$$

This formulation demonstrates that reinsertion distance effectively captures the essence of P-Permutation problems.

IV. Permutation Mutation Operators

We consider several common permutation mutation operators, widely used for GA mutation as well as within other metaheuristics [1], [10], [38]–[41]. Wherever a pair of random indices is required, we generate them as in Fig. 3.

Adjacent Swap (AdjSwap): The AdjSwap operator exchanges the positions of a random pair of adjacent elements.

Swap: The Swap operator chooses two different random permutation indices; and exchanges the chosen elements.

Insertion: Insertion removes a random element, and reinserts it at a different random location, shifting the elements between the removal and reinsertion points by 1 position.

Algorithm: GetRandomIndexPair(p)
Inputs: A permutation, p, of length N.
$a \leftarrow$ random integer from $\{1,\ldots,N\}$
$b \leftarrow$ random integer from $\{1,\ldots,(N-1)\}$
if $b \geq a$ **then** Increment b by 1.
if $b < a$ **then** Exchange a and b such that b holds the higher index.
return (a,b)

Fig. 3. Generates a random index pair (indices begin at 1).

Algorithm: Scramble(p)
Inputs: A permutation, p
$a,b \leftarrow$ **GetRandomIndexPair**(p) {Note: $b > a$}
changed $\leftarrow$ **false**
for $i = b$ **downto** $(a+2)$ **do**
  $j \leftarrow$ random integer from $\{a,\ldots,i\}$
  if $i \neq j$ **then**
    Swap the elements at positions i and j.
    changed $\leftarrow$ **true**
if (**not** changed) **or** (random Boolean is **true**) **then**
  Swap the elements at positions a and $(a+1)$.

Fig. 4. Pseudocode for the Scramble mutation operator.

Reversal: The Reversal operator, also known as Inversion, chooses two different random indices; and reverses the sub-permutation identified by those indices, inclusive.

Scramble: The Scramble operator picks two different random indices; and randomizes the identified sub-permutation. Most implementations allow the permutation to remain unchanged (e.g., to ease implementation). However, in our implementation, all possible reorderings, exclusive of the original, are equally likely, as seen in the pseudocode in Fig. 4.

Table I summarizes the local neighborhood size and runtimes of the operators. For AdjSwap, each permutation has $N-1$ neighbors, where N is the permutation length. For Insertion, there are $N(N-1)$ neighbors, since there are N options for the first index, and $N-1$ options for the second. Swap and Reversal generate half as many neighbors since the order the indices are chosen does not matter. The Scramble neighborhood includes all possible reorderings; and thus there are $(N!-1)$ neighbors. AdjSwap and Swap are constant runtime operations, while the runtime of the others is in $O(N)$.

V. Fitness Distance Correlation and Haystack(δ, N) Landscapes

We now explore FDC for several "Permutation in a Haystack" landscapes. Table II, column 1, lists the specific landscapes, involving permutations of length 10. The findings should generalize to larger permutations, and we do use larger permutations in our experiments later in the paper. All but three of these landscapes have a single optimal solution. Haystack$(\delta_{\text{AcyclicEdge}}, 10)$ has two solutions (the target permutation and its reverse). Haystack$(\delta_{\text{CyclicEdge}}, 10)$ has 20 optimal solutions, including all 10 possible rotations of each of the target permutation and its reverse. Haystack$(\delta_{\text{CyclicRType}}, 10)$ has 10 optimal solutions–the 10 rotations of the target permutation.

As discussed earlier in Section II, the ideal distance measure for defining the topology of the fitness landscape and for computing FDC is an edit distance where the operator we

TABLE I
PROPERTIES OF VARIOUS MUTATION OPERATORS.

Operator	Number of Neighbors	Runtime Complexity
AdjSwap	$N-1$	$O(1)$
Swap	$N(N-1)/2$	$O(1)$
Insertion	$N(N-1)$	$O(N)$
Reversal	$N(N-1)/2$	$O(N)$
Scramble	$N!-1$	$O(N)$

are analyzing defines the edit operations. For the landscapes with a single optimal solution, we use the following distance measures. We use Kendall tau distance for the AdjSwap operator. Kendall tau is sometimes referred to as "bubble sort" distance as it corresponds to the number of adjacent swaps performed by bubble sort. It thus corresponds to an edit distance where the only edit operation is AdjSwap. Interchange distance is used for the Swap operator, as by definition it is an edit distance with Swap as the only edit operation. Reinsertion distance is likewise used for the Insertion operator.

Choosing a distance measure corresponding to the Reversal operator was a challenge. Computing reversal edit distance is an NP-Hard problem [17]. Typically, a "surrogate" distance measure is needed, however, the best available approximations are insufficient for this purpose [13]. Despite the exponential time requirements to compute reversal edit distance exactly, we have chosen to do so anyway, and use the following:

$$\delta_{\text{Reversal}}(p_1,p_2)=\begin{Bmatrix}\text{Minimum number of reversals}\\ \text{needed to transform } p_1 \text{ into } p_2.\end{Bmatrix}. \quad (12)$$

We compute this reversal edit distance between a reference permutation and all $N!$ permutations, generating a lookup table indexed by a mapping between permutations and the integers in $\{0,1,\ldots,(N!-1)\}$. We generate this lookup table via a breadth-first exhaustive enumeration outward from the reference permutation, for a total cost of $O(N!N^2)$. However, the average cost per permutation is simply $O(N^2)$.

For the Scramble operator, the "edit distance" is straightforward. The distance between two identical permutations is 0; and otherwise it is 1. Since it is possible to jump from any permutation to any other via a single Scramble, every permutation is a direct neighbor of every other, leading to:

$$\delta_{\text{Scramble}}(p_1,p_2)=\begin{cases}0 & \text{if } p_1=p_2\\ 1 & \text{otherwise.}\end{cases} \quad (13)$$

We use the concept of reversal independence [15] to account for the two optimal solutions to Haystack($\delta_{\text{AcyclicEdge}},10$); and for that landscape, we compute distance as:

$$\delta'(p_1,p_2)=\min\{\delta(p_1,p_2),\delta(p_1,p_2')\}, \quad (14)$$

where p_2' is the reverse of p_2.

We use the concept of cyclic independence [15] to account for the 10 optimal solutions to Haystack($\delta_{\text{CyclicRType}},10$); and for this landscape, we compute distance as:

$$\delta'(p_1,p_2)=\min_{i\in\{0,\ldots,9\}}\delta(p_1,p_2^i), \quad (15)$$

where p_2^i is p_2 rotated i positions circularly.

Both reversal and cyclic independence are used to account for the 20 optimal solutions of Haystack($\delta_{\text{CyclicEdge}},10$):

$$\delta'(p_1,p_2)=\min_{i\in\{0,\ldots,9\}}\{\min\{\delta(p_1,p_2^i),\delta(p_1',p_2^i)\}\}. \quad (16)$$

For a Haystack(δ_f,N) landscape, we compute FDC as:

$$\text{FDC}=\frac{\text{Cov}(\delta_f(p_N,p),\delta_d(p_N,p))}{\sigma_{\delta_f(p_N,p)}\sigma_{\delta_d(p_N,p)}}, \quad (17)$$

where δ_f and p_N are the Haystack optimization objective and target permutation (Section III), and δ_d is the natural distance for the mutation operator. The standard deviations $\sigma_{\delta_f(p_N,p)}$, $\sigma_{\delta_d(p_N,p)}$, and the covariance Cov are computed over all permutations p of length N to avoid challenges in accurately computing FDC by sampling permutation space (e.g., samples tend to be equidistant to the target permutation p_N since the expected number of fixed points in a random permutation is one [42], leading to insufficient distance variation). FDC calculation in this way is thus limited to small problems.

Table II summarizes the FDC for the various mutation operators and search landscapes. Where we find the strongest FDC, we should expect the corresponding operators to perform well for the given search landscape. Where we find weak (or no correlation), we should expect the corresponding operator to perform poorly, such as Scramble for any of the landscapes. For most of the search landscapes, there is at least one operator with strong correlation (above 0.7). However, in the case of Haystack($\delta_{\text{RType}},10$), the strongest correlation is only 0.459 (the Reversal operator); and the strongest correlation for Haystack($\delta_{\text{CyclicRType}},10$) is 0.642 (the Insertion operator).

VI. THE CALCULUS OF SEARCH LANDSCAPES

Though FDC can be a powerful tool in the analysis of a search landscape, it does not tell the whole story. For example, consider the rather hopeless optimization problem Haystack($\delta_{\text{Scramble}},N$), where other than the single optimal solution, the landscape is perfectly flat regardless of our search operator. The search landscape associated with the Scramble operator has FDC equal to 1. The search landscapes associated with the other operators all have FDC rapidly approaching 0 as N increases. Scramble appears an excellent choice, and yet operator choice does not matter since the best we can do is wander randomly and any operator accomplishes that.

Consider this example where we must minimize a function, $f(x)$, of one real-valued variable $x\in[-N,N]$ of the form:

$$\begin{aligned}f(x)=&\lceil|x|\rceil(c-(-1)^{\lceil|x|\rceil}r)\\&-(\lceil|x|\rceil-|x|)(c-(-1)^{\lceil|x|\rceil}r(2\lceil|x|\rceil-1)).\end{aligned} \quad (18)$$

Fig. 5 illustrates this fitness landscape for $c=5$, $r=3$, and $N=100$. Assuming a neighborhood operator that makes small changes to the candidate solution x' (e.g., Gaussian mutation), the FDC of this search landscape is 0.822, computed using 20001 equally spaced values of x in the interval: $-100\le x\le 100$. Though it has very strong FDC, this search landscape is plagued by large numbers of deep local minima requiring steep challenging climbs for the search to escape.

To complement FDC, we additionally examine the rate of change of solution fitness within the local neighborhood of

TABLE II
FITNESS DISTANCE CORRELATION FOR COMMON PERMUTATION MUTATION OPERATORS AND HAYSTACK$(\delta, 10)$ SEARCH LANDSCAPES.

Search Landscape	# Optimal	AdjSwap	Swap	Insertion	Reversal	Scramble	Strongest Correlations	Other Strong Correlations
Haystack$(\delta_{\text{EM}}, 10)$	1	0.328	0.766	0.301	0.173	0.005	Swap	
Haystack$(\delta_{\text{Interchange}}, 10)$	1	0.241	1.000	0.182	0.199	0.003	Swap	
Haystack$(\delta_{\text{Dev2}}, 10)$	1	0.931	0.395	0.650	0.086	0.002	AdjSwap	Insertion
Haystack$(\delta_{\text{AcyclicEdge}}, 10)$	2	0.194	0.130	0.282	0.760	0.005	Reversal	
Haystack$(\delta_{\text{CyclicEdge}}, 10)$	20	0.382	0.380	0.477	0.841	0.014	Reversal	
Haystack$(\delta_{\text{RType}}, 10)$	1	0.085	0.009	0.422	0.459	0.005	Reversal, Insertion	
Haystack$(\delta_{\text{CyclicRType}}, 10)$	10	0.342	0.296	0.642	0.558	0.014	Insertion, Reversal	
Haystack$(\delta_{\tau}, 10)$	1	1.000	0.241	0.704	0.067	0.002	AdjSwap	Insertion
Haystack$(\delta_{\text{Reinsertion}}, 10)$	1	0.704	0.182	1.000	0.136	0.003	Insertion	AdjSwap

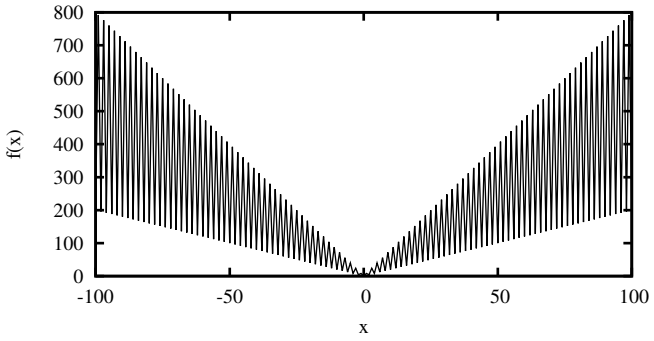


Fig. 5. Search landscape with strong FDC and numerous deep local minima.

a point on the fitness landscape. We adopt the following notation. First, let $\eta(p)$ be the set of all direct neighbors of point p on the landscape given the mutation operator under analysis. Now define the average rate of change of the fitness function, f, from p to neighbors, $p' \in \eta(p)$, as:

$$\Delta[f](p) = \frac{1}{|\eta(p)|} \sum_{p' \in \eta(p)} |f(p) - f(p')|. \tag{19}$$

For example, consider the OneMax problem with bit-strings of length N. Assuming bit-flip mutation, there are N neighbors (e.g., flipping each of the N bits). Each has fitness either 1 greater than, or 1 less than, the current candidate solution. Thus, $\Delta[f](p) = 1$. Now consider the same OneMax problem, but using an inversion operator (i.e., reversing a randomly selected sub-string). In this case, $\Delta[f](p) = 0$ since the number of one-bits does not change under an inversion. In the case of the fitness landscape of Fig. 5, $\Delta[f](x)$ is proportional to $|c - (-1)^{\lceil |x| \rceil} r(2\lceil |x| \rceil - 1)|$ which is simply the magnitude of the slope of the line upon which the point $(x, f(x))$ lies.

For Haystack(δ, N), where the fitness function is a distance metric δ, we equivalently define the average rate of change as:

$$\Delta[\delta](p) = \frac{1}{|\eta(p)|} \sum_{p' \in \eta(p)} \delta(p, p'). \tag{20}$$

We denote by $\Delta[f]$ the average of $\Delta[f](p)$ over all points p; and by $\Delta[\delta]$ the average of $\Delta[\delta](p)$ over all permutations p. For the landscape of Fig. 5, $\Delta[f]$ is approximately proportional to rN (i.e., $\Delta[f]$ grows linearly with problem size, N).

To provide a common mode of interpretation, we require that the maximum of the fitness function $f(p)$ ($\delta(p_1, p_2)$ in the case of Haystack(δ, N) landscapes) is in $O(N)$ (i.e., the range of possible fitness values must be proportional to the encoding length, N). To meet this requirement, we use the original version of deviation distance, Dev1, rather than Dev2; and we scale Kendall tau distance, which otherwise has a maximum of $N(N-1)/2$, to a maximum of N as follows:

$$\delta_{\tau'}(p_1, p_2) = \tfrac{2}{N-1} \delta_{\tau}(p_1, p_2). \tag{21}$$

Under this requirement, we have the following three cases:

$$\lim_{N \to \infty} \Delta[f] = 0, \tag{22a}$$

$$\lim_{N \to \infty} \Delta[f] = C \text{ for constant } C > 0, \text{ and} \tag{22b}$$

$$\lim_{N \to \infty} \Delta[f] = \infty. \tag{22c}$$

In the first case (22a), the search landscape is locally flat—e.g., the OneMax problem with an inversion operator. Haystack$(\delta_{\text{Scramble}}, N)$ with any operator is a permutation example. Regardless of permutation operator, $\Delta[f] = 0$ for this landscape. $\Delta[f](p) = 0$ for only some p in the landscape corresponds to local plateaus. When $\Delta[f] = 0$ search progress will be slow since escape from large flat plateaus is necessary.

In the second case (22b), the local steepness of the search landscape is indicated by C, with lower values for smooth gradual slopes. If stuck in the attraction basin of a local optima, it should be easier to escape for lower C values.

The last case (22c) corresponds to a search landscape with large local changes in fitness. This occurs for search landscapes with large numbers of deep local optima, such as our hypothetical search landscape of Fig. 5. Searching locally on such a landscape may be especially difficult.

Table III provides the $\Delta[\delta]$ for various search landscapes formed by combination of search operator and Haystack(δ, N) fitness function, computed in general in terms of permutation length N and not for any specific value of N. For each Haystack landscape, the last column of Table III lists the operators for which $\lim_{N \to \infty} \Delta[f] = C$, sorted by C. Unlike FDC, the $\Delta[\delta]$ derivations are not limited by problem size. Complete derivations of these are found in the Appendix.

VII. EXPERIMENTS WITH SIMULATED ANNEALING

To test our theoretical framework's effectiveness at predicting local search operator performance, we conducted experiments on the Haystack(δ, N) problem using SA. We chose SA because its random neighbor selection and ability to

TABLE III
AVERAGE LOCAL FITNESS RATE OF CHANGE, $\Delta[\delta]$, FOR VARIOUS HAYSTACK FITNESS FUNCTIONS, δ, AND SEARCH OPERATORS.

$\Delta[\delta]$	AdjSwap	Swap	Insertion	Reversal	Scramble	Operators where $\lim_{N\to\infty}\Delta[f]=C$ sorted increasing by C
$\Delta[\delta_{\text{EM}}]$	2	2	$\frac{N+4}{3}$	$\left[\frac{N+1}{3},\frac{N+4}{3}\right]$	$\frac{N+1}{3}$	AdjSwap $\sim$ Swap
$\Delta[\delta_{\text{Interchange}}]$	1	1	$\frac{N+1}{3}$	$\left\lfloor\frac{N+4}{6}\right\rfloor$	$\Delta[\delta]\geq\frac{N+4}{3}-\ln(\frac{N+4}{3})-\gamma$	AdjSwap $\sim$ Swap
$\Delta[\delta_{\text{Dev1}}]$	0	2/3	2/3	$\left(\frac{N+6}{12},\frac{N+22}{12}\right]$	$\left(\frac{N+6}{18},\frac{N+16}{18}\right]$	Swap $\sim$ Insertion
$\Delta[\delta_{\text{AcyclicEdge}}]$	2	4	3	2	$\frac{N+1}{3}$	AdjSwap $\sim$ Reversal $\prec$ Insertion $\prec$ Swap
$\Delta[\delta_{\text{CyclicEdge}}]$	2	4	3	2	$\frac{N+1}{3}$	AdjSwap $\sim$ Reversal $\prec$ Insertion $\prec$ Swap
$\Delta[\delta_{\text{RType}}]$	3	4	3	$\frac{N+7}{3}$	$\left[\frac{N}{3}+\frac{5}{6},\frac{N+4}{3}\right)$	AdjSwap $\sim$ Insertion $\prec$ Swap
$\Delta[\delta_{\text{CyclicRType}}]$	3	4	3	$\frac{N+7}{3}$	$\left[\frac{N}{3}+\frac{5}{6},\frac{N+4}{3}\right)$	AdjSwap $\sim$ Insertion $\prec$ Swap
$\Delta[\delta_{\tau'}]$	0	4/3	2/3	$\left(\frac{N+4}{6},\frac{N+10}{6}\right]$	$\left(\frac{N+4}{12},\frac{N+10}{12}\right]$	Insertion $\prec$ Swap
$\Delta[\delta_{\text{Reinsertion}}]$	1	2	1	$\frac{N+1}{3}$	$\frac{N+4}{3}-2\sqrt{\frac{N+4}{3}}+1.77\sqrt[6]{\frac{N+4}{3}}$	AdjSwap $\sim$ Insertion $\prec$ Swap

accept weak neighbors are similar to GA mutation; and it is free of population-related parameters requiring tuning (e.g., mutation rate and population size). Thus, it enables focusing on the effects of mutation without regard to control parameter sensitivity. For this same reason, we use the parameter-free Modified Lam annealing schedule [43], [44], which dynamically adapts the temperature parameter using search feedback.

A well known theoretical result states that if the annealing schedule "cools" the temperature at a sufficiently slow rate to maintain the system in a state of thermal equilibrium, then SA converges to the globally optimal solution with probability 1. However, this result requires such a slow cooling rate that a prohibitively long SA run is needed to settle upon a solution.

Lam and Delosme studied the behavior of SA under an approximate thermal equilibrium model, D-equilibrium, that balances computation time and solution quality [45]. They showed that the ideal run of SA accepts neighbors at a rate of 0.44; and they devised an annealing schedule that tracks this acceptance rate. Lam and Delosme's SA uses a monotonically decreasing temperature schedule, and adjusts the neighborhood function to maintain the acceptance rate as near 0.44 as possible. They increase the size of the local neighborhood to decrease the acceptance rate; and decrease the size of the local neighborhood to increase the acceptance rate. Their motivation is the assumption that nearby search states are of similar quality; and thus, a smaller local neighborhood implies smaller difference between current fitness and neighbor fitness, which leads to higher probability of neighbor acceptance.

Swartz later rigorously studied the behavior of Lam and Delosme's annealing schedule [43]. He observed that at the beginning of the search, the acceptance rate is near 1.0 (i.e., all random moves accepted) and decreases at an exponential rate during the first 15% of the run when it reaches the target acceptance rate of 0.44. For the next 50% of the run, the acceptance rate remains nearly constant, when it then begins an exponential decline for the remainder of the run. Swartz then proposed an alternative approach to tracking the theoretically ideal acceptance rate. Rather than adjusting the size of the local neighborhood, Swartz's Modified Lam schedule varies the temperature—increasing temperature to increase acceptance rate, and decreasing temperature to decrease acceptance rate. Boyan suggested a specific implementation of Swartz's Modified Lam schedule [44], which we here adopt and which is found in Fig. 6. We have used this SA with excellent results in some of our own prior work (e.g., [40]).

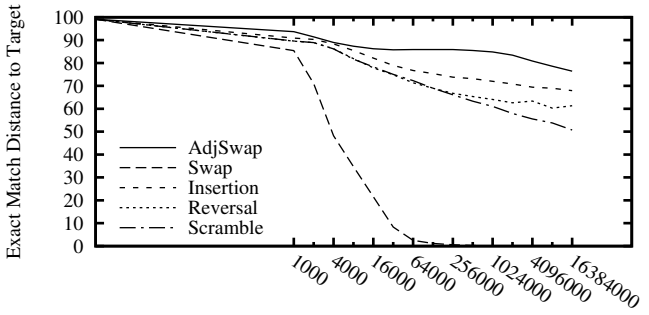
Algorithm: Modified Lam Annealing
$S \leftarrow$ GenerateRandomInitialState
$T \leftarrow 0.5$
AcceptRate $\leftarrow 0.5$
for $i = 1$ **to** MaxEvals **do**
  $S' \leftarrow$ random selection from $\eta(S)$
  if Cost(S') $\leq$ Cost(S) **or** Rand $\in [0,1) < e^{(\text{Cost}(S)-\text{Cost}(S'))/T}$ **then**
    $S \leftarrow S'$
    AcceptRate $\leftarrow \frac{1}{500}(499 \cdot \text{AcceptRate} + 1)$
  else AcceptRate $\leftarrow \frac{1}{500}(499 \cdot \text{AcceptRate})$
  if $i/\text{MaxEvals} < 0.15$ **then**
    LamRate $\leftarrow 0.44 + 0.56 \cdot 560^{-i/\text{MaxEvals}/0.15}$
  else if $0.15 \leq i/\text{MaxEvals} < 0.65$ **then**
    LamRate $\leftarrow 0.44$
  else if $0.65 \leq i/\text{MaxEvals}$ **then**
    LamRate $\leftarrow 0.44 \cdot 440^{-(i/\text{MaxEvals}-0.65)/0.35}$
  if AcceptRate $>$ LamRate **then** $T \leftarrow 0.999 \cdot T$
  else $T \leftarrow T/0.999$
return S

Fig. 6. SA with the Modified Lam annealing schedule.

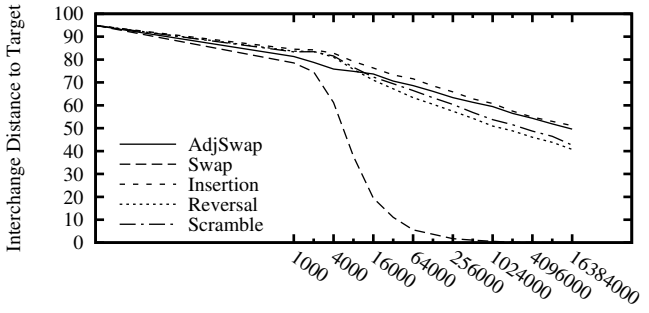
A. Experimental Setup

In our Haystack(δ, N) experiments, we use a permutation length, $N = 100$. We consider run lengths (in SA evaluations): $1000 \cdot 2^i$ for $i \in \{0, 1, \ldots, 14\}$. For each combination of search landscape, neighborhood function, and run length, we execute 100 runs of the Modified Lam SA (Fig. 6). To ensure that all neighborhood operators face instances of equivalent difficulty, we use a common set of 100 random permutations of the integers in $\{0, 1, \ldots, (N-1)\}$ as the random initial configurations. We compute paired sample t-tests for each pair of neighborhood operators at each run length.

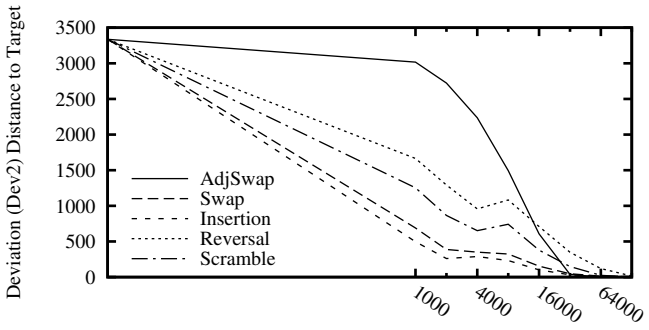
We used a Ubuntu 12.04 workstation, with an Intel i7 CPU running at 3.4GHz with 16GB RAM. All code is implemented in Java and compiled with OpenJDK 8.



(a)

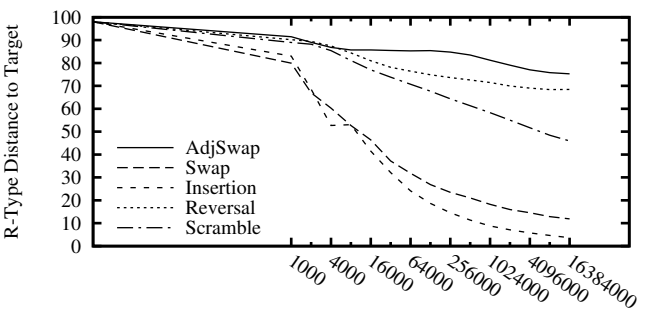


(b)

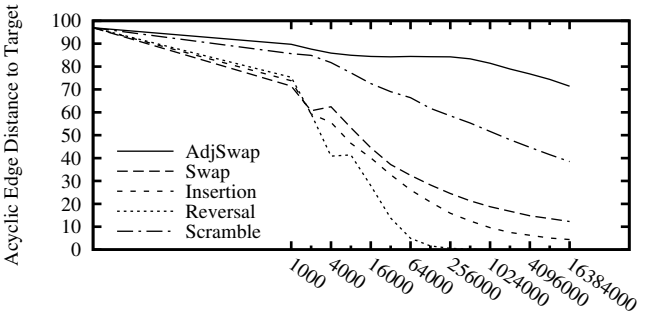


(c)

Fig. 7. SA results (100 run averages): (a) Haystack($\delta_{\text{EM}}, 100$), (b) Haystack($\delta_{\text{Interchange}}, 100$), (c) Haystack($\delta_{\text{Dev2}}, 100$).



(a)



(b)

Fig. 8. SA results (100 run averages): (a) Haystack($\delta_{\text{RType}}, 100$), (b) Haystack($\delta_{\text{AcyclicEdge}}, 100$).

B. A-Permutation Results

The A-Permutation results are in Fig. 7, with run length on the x-axis at log scale, and fitness on the y-axis. FDC (Table II) correctly predicts Swap will dominate for Haystack($\delta_{\text{EM}}, 100$) and Haystack($\delta_{\text{Interchange}}, 100$) as seen in Figs. 7a and 7b.

Although we have extremely strong FDC for AdjSwap and Haystack($\delta_{\text{Dev2}}, 100$), AdjSwap does quite poorly (Fig. 7c). AdjSwap's poor performance is explained by $\Delta[\delta]$, which converges to 0 with permutation length (Table III). The analysis of $\Delta[\delta]$ predicts that Swap and Insertion should perform best, as is confirmed in our SA experiments. Interestingly, though Insertion has strong FDC ($R = 0.65$), Swap has only moderate FDC ($R = 0.395$). Thus, if we rely solely on FDC, we may not consider Swap to be a good candidate for this search problem.

For runs of length 256000 or less on the Haystack($\delta_{\text{EM}}, 100$) problem, there is no statistical significance between the performance of Reversal and Scramble. However, for all other pairs of operator at all run lengths for this landscape, paired sample t-tests find extremely significant results (p-values no higher than 0.00019 and in some cases lower than 10^{-100}). Significance of the Haystack($\delta_{\text{Interchange}}, 100$) results are similar. There is little to no significance for the difference between Reversal and Scramble, but otherwise all results are extremely statistically significant (with the exception of AdjSwap and Insertion for very long runs). For Haystack($\delta_{\text{Dev2}}, 100$), performance differences between pairs of operator are extremely statistically significant except near the very end of runs (e.g., see where the AdjSwap curve crosses the others).

C. R-Permutation Results

Fig. 8 shows the results for the R-Permutation landscapes: R-type and acyclic edge distance. We show only the results for the acyclic forms, as the cyclic results are nearly identical.

For the R-type landscape, Reversal and Insertion are most promising based on FDC. Insertion is in fact the best performer (Fig. 8a), while Reversal does quite poorly. The poor performance of Reversal, despite moderately strong FDC, is related to our earlier example search landscape in Fig. 5. The R-type landscape has directed edges, so although Reversal can replace two bad edges with two better edges, it also reverses the direction on a number of edges that grows linearly with permutation length. In order for Reversal to make a small amount of progress, it must climb out of deep local minima. Our analysis of $\Delta[\delta]$ detects this—for Reversal, $\lim_{N\to\infty} \Delta[\delta] = \infty$. Using the analysis of $\Delta[\delta]$, we predict

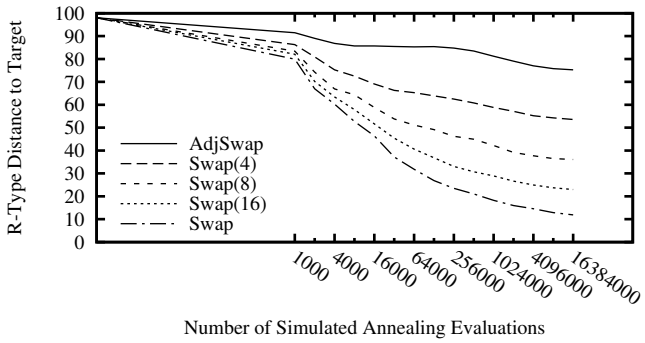


Fig. 9. SA with window-limited Swap (100 run averages) on Haystack$(\delta_{\text{RType}}, 100)$.

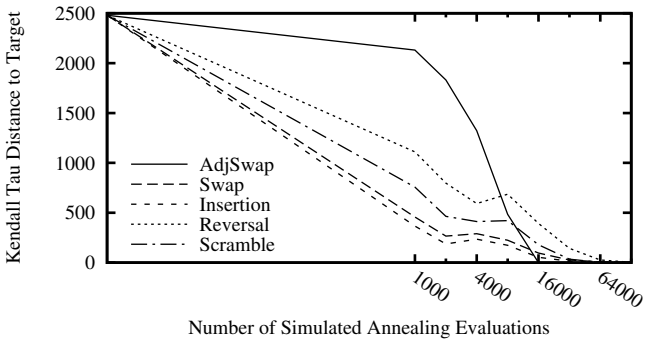
Insertion would perform best, followed by Swap, which is exactly what the results show (Fig. 8a). Note that the $\Delta[\delta]$ analysis does not require the costly reversal edit distance computation, required for FDC; and yet $\Delta[\delta]$ exhibits stronger prescriptive power. The R-type results for all pairs of operator at all run lengths of more than 8000 evaluations are extremely significant (p-values from 10^{-18} to lower than 10^{-100}).

On the R-type landscape, $\Delta[\delta]$ also predicts AdjSwap will perform well when it actually performs the worst. AdjSwap likely converges slowly due to a small neighborhood relative to the others (Table I). AdjSwap replaces a small constant number of edges, as does Insertion and Swap; but it is limited to a much smaller pool of possible replacements leading to slow search progress. This is consistent with research findings on the effects of neighborhood size on search performance (e.g., [46]–[48]). To test this hypothesis, we consider a window-limited variant of Swap, Swap(L), that swaps a random pair of elements at most L positions apart [16]. AdjSwap is equivalent to Swap(1); and Swap is equivalent to Swap(∞). L controls the size of the neighborhood, defining a continuum of operators from AdjSwap to Swap. In Fig. 9, we see that search performance improves as L increases.

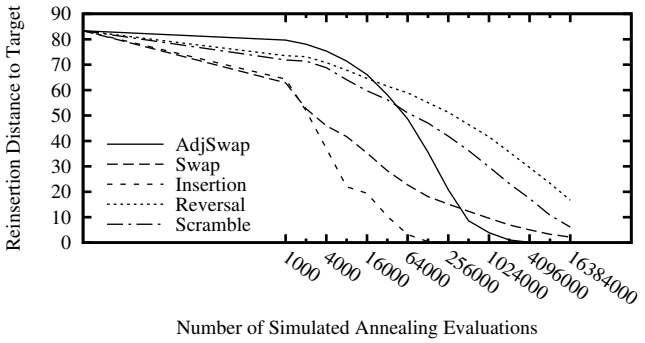
FDC predicts Reversal as the dominant operator for edge distance landscapes ($R = 0.76$ and $R = 0.84$). SA confirms this (Fig. 8b), yet Insertion and Swap are not far behind despite much lower FDC. The $\Delta[\delta]$ analysis predicts the very relative performance of these three operators; but also predicts AdjSwap would do as well as Reversal, when it actually performs the worst, again due to its small neighborhood. The edge distance results for all pairs of operator at all run lengths of more than 2000 evaluations are extremely significant (p-values from 10^{-15} to less than 10^{-100}).

D. P-Permutation Results

Fig. 10 shows the P-Permutation results. AdjSwap's perfect FDC for Haystack$(\delta_\tau, 100)$ suggests it should dominate. Insertion also exhibits strong FDC. For shorter runs, AdjSwap does poorly relative to all of the other operators, however, for longer runs, it quite dramatically surpasses the others and is the first to converge to the optimal solution for all 100 runs (Fig. 10a). The very slow convergence early in the run is explained by $\lim_{N\to\infty} \Delta[\delta] = 0$, which means the landscape



(a)



(b)

Fig. 10. SA results (100 run averages): (a) Haystack$(\delta_\tau, 100)$, (b) Haystack$(\delta_{\text{Reinsert}}, 100)$.

is extremely flat. AdjSwap is essentially wandering about on a large plateau. The analysis of $\Delta[\delta]$ shows that Insertion, followed by Swap, should perform well, as is confirmed by SA. All performance differences between pairs of operator are extremely statistically significant except near the very end of runs (e.g., where the AdjSwap curve crosses the others).

Insertion's perfect FDC suggests it as the likely dominant operator for Haystack$(\delta_{\text{Reinsert}}, 100)$. AdjSwap also has strong FDC ($R = 0.7$), and is the second operator to enable SA to converge to the optimal (Fig. 10b). This is consistent with the analysis of $\Delta[\delta]$ (Table III). Interestingly, for shorter SA runs, Swap performs at a level between Insertion and AdjSwap, which is explained by the low constant value of $\Delta[\delta]$. The Haystack$(\delta_{\text{Reinsert}}, 100)$ results for all pairs of operators at all run lengths of more than 2000 evaluations are extremely significant (p-values from 10^{-10} to less than 10^{-100}), except where the AdjSwap curve crosses the others.

VIII. Conclusion

In this paper, we have presented the *Permutation in a Haystack*, a new optimization problem enabling effective study of the behavior and performance of local neighborhood, mutation, and crossover operators on a wide variety of permutation search landscapes. We have provided a collection of permutation distance metrics that spans the three major categories of permutation search problem (A-Permutation, R-Permutation, and P-Permutation), and enables parameterizing the Haystack(δ, N) problem by permutation problem class.

We have also proposed a *Calculus of Search Landscapes*, focusing on local rates of change of fitness in analyzing the local behavior of neighborhood operators on the landscape. We saw examples of search landscapes that exhibit strong FDC, but which either contain large flat expanses or are plagued by large numbers of deep local optima. The search landscape calculus can detect both of these characteristics; and can do so without requiring computation of the natural distance associated with the search operator. Computing FDC requires either calculation of the operator's natural edit distance, which can be cost prohibitive for some operators (e.g., Reversal), or selection of an appropriate surrogate distance which has its own challenges. The search landscape calculus does not suffer from these and other problem size limitations.

To validate our approach, and to demonstrate its prescriptive power, we conducted experiments with SA on Haystack(δ, N) problems encompassing all three permutation problem categories. In all cases, the search landscape calculus predicted the dominant mutation operators for the landscape, and even the relative order of performance on landscapes where multiple promising operators were identified. Though it was not the primary objective of our study, the results provide guidance on neighborhood operator selection for permutation problems. For example, Swap is the clear preferred operator for most A-Permutation problems, with Insertion appropriate when small positional deviations imply similar solutions. Reversal is best for R-Permutation problems with undirected edges, but is far too disruptive when edges are directed; while Insertion and Swap are both promising for directed edges. The dominant operators for P-Permutation problems are Insertion, Swap, and AdjSwap (the latter for longer runs).

Our experimental results on the Haystack(δ, N) problem provide insight beyond SA to other metaheuristics, including those that are population-based such as the GA; and are also consistent with applications to more specific problems found in the literature. For example, there are numerous examples available of the strength of Swap for A-Permutation problems, such as for GA mutation for graph isomorphism [49] and within a local hill-climber for the largest common subgraph problem [50]—both involve optimizing a node mapping—as well as other absolute ordering problems such as in a GA for Suduko solving [51]. Reversal mutation is often found in GAs for R-Permutation problems, such as vehicle routing (e.g., [52]), except when edges are directed, such as scheduling with sequence-dependent setup constraints, where you typically find Insertion mutation (e.g., [10], [40]). The Haystack(δ, N) problem enables effectively isolating the permutation properties of interest, and the search landscape calculus offers a formal approach to gaining a better understanding of search operator behavior in a problem independent manner.

It is our plan to continue to expand our work into the study of search landscapes for other representations, as well as for more complex permutation operators (e.g., crossover).

APPENDIX
A-PERMUTATION $\Delta[\delta]$ DERIVATIONS

In our subsequent analysis, we will find the following useful. Let X be the number of elements in the sub-permutation induced by the random index selection. The expected number of elements, $E[X]$, in a random sub-permutation is:

$$E[X] = \frac{\sum_{i=1}^{N-1}(N-i)(i+1)}{\sum_{i=1}^{N-1}(N-i)} = \frac{N+4}{3}, \quad (23)$$

since there are $(N-i)$ index pairs, i positions apart, each of which is an $(i+1)$ element sub-permutation. The variance is:

$$\text{Var}[X] = \frac{\sum_{i=1}^{N-1}(N-i)(i+1)^2}{\sum_{i=1}^{N-1}(N-i)} - (\frac{N+4}{3})^2 = \frac{N^2-N-2}{18}. \quad (24)$$

Exact Match

AdjSwap and Swap: Both AdjSwap and Swap always exchange exactly two elements. Therefore, $\Delta[\delta_{\text{EM}}] = 2$.

Insertion: All of the elements in the sub-permutation induced by the selection of the random indices change locations. Thus, $\Delta[\delta_{\text{EM}}] = (N+4)/3$, computed earlier in (23).

Reversal: For simplicity, we bound $\Delta[\delta_{\text{EM}}]$ for Reversal rather than an exact calculation. First, if a k-element sub-permutation is inverted, k elements change locations if k is even, and $k-1$ otherwise. Since a random sub-permutation has $(N+4)/3$ elements on average, we bound $\Delta[\delta_{\text{EM}}]$ with:

$$\frac{N+1}{3} \leq \Delta[\delta_{\text{EM}}] \leq \frac{N+4}{3}. \quad (25)$$

Scramble: The expected number of fixed points in a random permutation is 1 [42]. The expected length of a randomly selected sub-permutation is $(N+4)/3$. Therefore, $\Delta[\delta_{\text{EM}}]$ for Scramble neighbors is: $(N+4)/3-1 = (N+1)/3$.

Interchange Distance

AdjSwap and Swap: $\Delta[\delta_{\text{Interchange}}]$ for Swap or AdjSwap is 1 (by operator definition).

Insertion: The sub-permutation affected by an Insertion on average consists of $(N+4)/3$ elements, inducing a permutation cycle with the same number of elements. Removing a permutation cycle of k elements requires $k-1$ interchanges. Thus, $\Delta[\delta_{\text{Interchange}}]$ is $(N+1)/3$.

Reversal: When reversing a k-element sub-permutation, the i-th elements from the start and end of the reversal are exchanged, forming a 2-cycle. There are $\lfloor k/2 \rfloor$ such 2-cycles, plus one fixed point when k is odd. The elements outside the reversal form $N-k$ singleton cycles, for a total of: $N-k+\lceil k/2 \rceil$ cycles. From (3), we have $\delta_{\text{Interchange}} = N-(N-k+\lceil k/2 \rceil) = \lfloor k/2 \rfloor$. Since the expected number of elements in the sub-permutation is $(N+4)/3$, $\Delta[\delta_{\text{Interchange}}]$ is $\lfloor (N+4)/6 \rfloor$.

Scramble: The expected number of cycles of a randomized permutation of k elements is H_k (the k-th Harmonic number) [42]. We use an approximation $H_k \sim \ln(k) + \gamma$ where $\gamma \sim 0.5772156649$ is Euler's constant [42]. Since the expected number of elements in a random sub-permutation is $(N+4)/3$, the randomized sub-permutation contributes approximately $\ln((N+4)/3) + \gamma$ permutation cycles. There are an additional $N-(N+4)/3$ singleton cycles (elements outside the random sub-permutation). Therefore, $\Delta[\delta_{\text{Interchange}}]$ is approximately:

$$\Delta[\delta_{\text{Interchange}}] \geq \frac{N+4}{3} - \ln(\frac{N+4}{3}) - \gamma. \quad (26)$$

Note that the $\geq$ is due to an approximation of the logarithmic term relying on $E[\ln(X)] \leq \ln(E[X])$.

TABLE IV
AVERAGE LOCAL FITNESS RATE OF CHANGE, $\Delta[\delta]$, FOR VARIOUS HAYSTACK FITNESS FUNCTIONS, δ, AND SEARCH OPERATORS.

$\Delta[\delta]$	AdjSwap	Swap	Insertion	Reversal	Scramble
$\Delta[\delta_{\text{Dev2}}]$	2	$\frac{2N+2}{3}$	$\frac{2N+2}{3}$	$\frac{N^2+5N+10}{12}$	$\frac{N^2+5N+4}{18}$
$\Delta[\delta_{\tau}]$	1	$\frac{2N-1}{3}$	$\frac{N+1}{3}$	$\frac{N^2+3N+2}{12}$	$\frac{N^2+3N+2}{24}$

Deviation Distance

Our earlier search landscape analysis of Section VI relied specifically on the original version of deviation distance, Dev1, since the maximum value of Dev1 is proportional to N. In deriving the $\Delta[\delta_{\text{Dev1}}]$ values seen earlier in Table III, we begin by deriving the results for the modified version of deviation distance, Dev2, which are summarized in Table IV.

AdjSwap: AdjSwap always exchanges two adjacent elements, causing a positional displacement of 1 each. Therefore, the Dev2 distance from an AdjSwap mutant to the original permutation is 2. For Ronald's version, Dev1, we divide by $N-1$, which in the limit for large N converges to 0.

Swap: Recall that there are $(N-i)$ pairs of indices exactly i positions apart. If two elements that are i positions apart are swapped, then the pair exhibit a total positional displacement of $2i$. Thus, $\Delta[\delta_{\text{Dev2}}]$ can be computed as follows:

$$\frac{\sum_{i=1}^{N-1}(N-i)2i}{\sum_{i=1}^{N-1}(N-i)} = \frac{2}{3}(N+1) \tag{27}$$

$\Delta[\delta_{\text{Dev1}}] = 2(N+1)/(3(N-1))$, which converges to $2/3$.

Insertion: If the removal and reinsertion points are i positions apart, i elements are displaced 1 position each and 1 element is displaced i positions, for a total displacement of $2i$. Therefore, $\Delta[\delta_{\text{Dev2}}]$ is the same as in (27).

Reversal: If we reverse a k-element sub-permutation, the distance between the original and the mutant permutations depends upon whether k is odd or even. When k is odd, the middle element does not move, 2 elements move 2 positions each, 2 elements move 4 positions each, etc. This leads to $\Delta[\delta_{\text{Dev2}}]$, for odd k, equal to:

$$\sum_{i=1}^{(k-1)/2}(2i)2 = \sum_{i=1}^{(k-1)/2} 4i = \tfrac{1}{2}(k^2-1). \tag{28}$$

For even k, it can be computed in a similar way:

$$\sum_{i=1}^{k/2}(2i-1)2 = \sum_{i=1}^{k/2}(4i-2) = \tfrac{1}{2}k^2. \tag{29}$$

The distance induced by a k-element Reversal is expressed succinctly as $\lfloor k^2/2 \rfloor$. Since $E[X^2] = (E[X])^2 + \text{Var}[X]$, then from (23) and (24), $\Delta[\delta_{\text{Dev2}}]$ is computed as:

$$\frac{(\frac{N+4}{3})^2 + \frac{N^2-N-2}{18}}{2} = \frac{N^2+5N+10}{12}. \tag{30}$$

For Ronald's version, Dev1, we divide by $N-1$. $\Delta[\delta_{\text{Dev1}}]$ is bound as follows (assumes $N > 1$):

$$\frac{N+6}{12} < \frac{N+6+\frac{16}{N-1}}{12} = \frac{N^2+5N+10}{12(N-1)} = \Delta[\delta_{\text{Dev1}}] \leq \frac{N+22}{12}. \tag{31}$$

Scramble: First consider the expected displacement of the i-th element of a scrambled k-element sub-permutation. Element i can occur in each of the k locations with equal probability, thus the expected displacement of element i is:

$$\tfrac{1}{k}\sum_{j=1}^{(i-1)}(i-j) + \tfrac{1}{k}\sum_{j=i+1}^{k}(j-i) = \tfrac{2i^2-2i-2ik+k+k^2}{2k}. \tag{32}$$

The expected sum of displacements of the k elements is thus:

$$\sum_{i=1}^{k}\tfrac{2i^2-2i-2ik+k+k^2}{2k} = \tfrac{1}{3}(k^2-1). \tag{33}$$

We can thus compute $\Delta[\delta_{\text{Dev2}}]$ for Scramble as (employing $E[X^2] = (E[X])^2 + \text{Var}[X]$, (23), and (24)):

$$\frac{(\frac{N+4}{3})^2 + \frac{N^2-N-2}{18} - 1}{3} = \frac{N^2+5N+4}{18}. \tag{34}$$

We compute $\Delta[\delta_{\text{Dev1}}]$ by dividing (34) by $(N-1)$, yielding $(N^2+5N+4)/(18(N-1))$, which we bound as:

$$\frac{N+6}{18} < \frac{N+6+\frac{10}{N-1}}{18} = \frac{N^2+5N+4}{18(N-1)} = \Delta[\delta_{\text{Dev1}}] \leq \frac{N+16}{18}. \tag{35}$$

R-PERMUTATION $\Delta[\delta]$ DERIVATIONS

Edge Distance (Cyclic and Acyclic)

AdjSwap: Since swapping two adjacent elements replaces exactly 2 edges, $\Delta[\delta_{\text{CyclicEdge}}] = 2$. AdjSwap only replaces one edge in the acyclic case when the swap occurs at an end of the permutation. However, $\Delta[\delta_{\text{AcyclicEdge}}] = 2$ in the limit.

Swap: An adjacent swap, or when the swapped elements are separated by a single element, results in a neighbor a cyclic edge distance of 2 from the original ($2N$ such cases). All other swaps replace 4 edges ($(N^2-5N)/2$ such cases). Thus, due to the dominance of the latter cases, $\Delta[\delta_{\text{CyclicEdge}}]$ and $\Delta[\delta_{\text{AcyclicEdge}}]$ both converge to 4.

Insertion: In the limit for large permutations, an Insertion replaces 3 edges on average (2 edges from the removal point, plus a third at the reinsertion point).

Reversal: All reversals, other than a complete reversal, replace 2 edges (at the endpoints of the reversal), so (in the limit) $\Delta[\delta_{\text{CyclicEdge}}]$ and $\Delta[\delta_{\text{AcyclicEdge}}]$ both converge to 2.

Scramble: Let $\{a, b\}$ be two elements of a k-element sub-permutation. There are $2(k-1)(k-2)! = 2(k-1)!$ permutations of k elements such that a is adjacent to b, since there are $k-1$ possible locations for the pair of elements, two ways of ordering the pair, and $(k-2)!$ ways of filling in the remaining $(k-2)$ elements. Therefore, if a and b are adjacent before the scramble, the probability that they are still adjacent afterwards is: $2(k-1)!/k! = 2/k$. A k-element sub-permutation has $k-1$ adjacent pairs of elements. Therefore, the expected number of interior edges retained during a scramble of a k-element sub-permutation is: $2(k-1)/k$. The sub-permutation is connected to the rest of the permutation at its endpoints. The probability of retaining a connecting edge is $1/k$. The expected number of edges retained is therefore: $2(k-1)/k + 2/k = 2$. The expected number of edges lost is $(k+1)-2 = (k-1)$ since there are $k+1$ edges counting the 2 connecting edges. $\Delta[\delta_{\text{CyclicEdge}}]$ converges to $\Delta[\delta_{\text{AcyclicEdge}}]$ in the limit since the expected contribution to the distance of the

edge between the endpoints is negligible. Since the expected length of a random sub-permutation is $(N+4)/3$, $\Delta[\delta_{\text{CyclicEdge}}]$ and $\Delta[\delta_{\text{AcyclicEdge}}]$ are therefore $(N+1)/3$.

R-Type Distance

AdjSwap: The behavior of AdjSwap is similar for R-type distance as it was for edge distance. However, since the edges are now directed edges, AdjSwap also breaks the edge between the swapped pair, so $\Delta[\delta]$ for the R-type distances (both form) is one more than they were for edge distance.

Swap and Insertion: The $\Delta[\delta]$ for Swap and Insertion are the same as they were for edge distance.

Reversal: In the average case, a sub-permutation with $(N+4)/3$ elements is inverted which breaks the $(N+1)/3$ interior edges of that sub-permutation and also breaks the two edges connecting it to the rest of the permutation. Thus, for Reversal, $\Delta[\delta_{\text{RType}}]$ and $\Delta[\delta_{\text{CyclicRType}}]$ are both $(N+7)/3$.

Scramble: If one scrambles a k-element sub-permutation, the probability of retaining any given edge is $1/k$ which is half that of the edge distance case since the order of the adjacent pair matters. The probability of retaining each of the two edges connecting the sub-permutation to the rest is $1/k$. There are $k-1$ interior edges and 2 connecting edges. The expected number of retained edges is: $(k+1)/k$. Thus, the expected number of edges lost is: $(k+1)-(k+1)/k = (k^2-1)/k$. Substituting the expected length of a random sub-permutation for k and simplifying yields: $(N^2+8N+7)/(3N+12)$. We bound $\Delta[\delta_{\text{RType}}]$ and $\Delta[\delta_{\text{CyclicRType}}]$ as:

$$\tfrac{1}{3}N+\tfrac{5}{6} \leq \tfrac{N+4}{3}-\tfrac{3}{N+4} = \Delta[\delta] = \tfrac{N^2+8N+7}{3N+12} < \tfrac{N+4}{3}. \quad (36)$$

P-PERMUTATION $\Delta[\delta]$ DERIVATIONS

Kendall Tau Distance

We begin by deriving $\Delta[\delta_\tau]$ for the original form of Kendall tau distance, which we have summarized in Table IV. We use these to derive the $\Delta[\delta_{\tau'}]$ values seen earlier in Table III.

AdjSwap: AdjSwap always affects exactly 1 element precedence relation, and thus an AdjSwap neighbor is always a Kendall tau distance of 1 from the original permutation.

Swap: If the endpoints of a sub-permutation with k elements are swapped, the pairwise rank orders of $(k-1)+(k-2) = 2k-3$ elements are changed. Since a random sub-permutation has an expected number of elements equal to $(N+4)/3$, the $\Delta[\delta_\tau]$ for a Swap is computed as: $2(N+4)/3-3 = (2N-1)/3$.

Insertion: On average, $(N+1)/3$ pairwise rankings (with the removed element) are changed during an Insertion.

Reversal: To compute $\Delta[\delta_\tau]$ for a Reversal, first observe that if a sub-permutation with k elements is reversed, that all $k(k-1)/2 = (k^2-k)/2$ pairwise rankings within that sub-permutation are changed. Employing $E[X^2] = (E[X])^2 + \text{Var}[X]$, (23), and (24), we can compute $\Delta[\delta_\tau]$ as:

$$\frac{(\frac{N+4}{3})^2+\frac{N^2-N-2}{18}-\frac{N+4}{3}}{2} = \frac{N^2+3N+2}{12} \quad (37)$$

Scramble: For $\Delta[\delta_\tau]$, first consider that if you randomize a k-element permutation such that all possible orderings are equally likely, there is an equal probability that element x will rank higher than element y as there is that element x will rank lower than element y. Thus, for any such pair there is a probability of 0.5 that the relative ranks will change. Since there are $k(k-1)/2$ such pairings in a k-element sub-permutation, the expected number of rank reversals is $k(k-1)/4$. This is half the expected Kendall tau distance induced by a reversal, and thus $\Delta[\delta_\tau]$ for Scramble is: $(N^2+3N+2)/24$.

To obtain $\Delta[\delta_{\tau'}]$, we multiply $\Delta[\delta_\tau]$ by $2/(N-1)$. For AdjSwap, $\Delta[\delta_{\tau'}]$ converges to 0 as N increases. Swap's $\Delta[\delta_{\tau'}]$ converges to $4/3$. For Insertion, $\Delta[\delta_{\tau'}]$ converges to $2/3$. For Reversal, we compute the following bounds (for $N>1$):

$$\tfrac{N+4}{6} < \tfrac{N+4}{6}+\tfrac{1}{N-1} = (\tfrac{N^2+3N+2}{12})(\tfrac{2}{N-1}) = \Delta[\delta_{\tau'}] \leq \tfrac{N+10}{6}. \quad (38)$$

For Scramble, the bounds for $\Delta[\delta_{\tau'}]$ are half those of Reversal.

Reinsertion Distance

AdjSwap: Since an AdjSwap neighbor can always be "edited" into the original permutation by a single removal/reinsertion operation, all AdjSwap neighbors are a reinsertion distance of 1 from the original.

Swap: An adjacent swap ($N-1$ such cases) leads to a reinsertion distance of 1; while for all other swaps, it is 2. Since the number of these latter cases is proportional to N^2, $\Delta[\delta_{\text{Reinsertion}}]$ for Swap neighbors converges to 2 in the limit.

Insertion: $\Delta[\delta_{\text{Reinsertion}}]$ for Insertion is 1 by definition.

Reversal: On average, a sub-permutation with $(N+4)/3$ elements is reversed, only one of which is on the longest non-contiguous subsequence; and thus, $(N+1)/3$ are not.

Scramble: The expected length of the longest non-contiguous common subsequence in random permutations of length k can be approximated with: $2\sqrt{k}-1.77108\sqrt[6]{k}$ [53]. The expected number of elements in the random sub-permutation is $(N+4)/3$, thus $\Delta[\delta_{\text{Reinsertion}}]$ is approximately:

$$\Delta[\delta_{\text{Reinsertion}}] \sim \tfrac{N+4}{3}-2\sqrt{\tfrac{N+4}{3}}+1.77108\sqrt[6]{\tfrac{N+4}{3}}. \quad (39)$$

REFERENCES

[1] V. Campos, M. Laguna, and R. Martí, "Context-independent scatter and tabu search for permutation problems," *INFORMS Journal on Computing*, vol. 17, no. 1, pp. 111–122, 2005.

[2] R. Martí, M. Laguna, and V. Campos, "Scatter search vs. genetic algorithms," in *Metaheuristic Optimization via Memory and Evolution*. Springer, 2005, pp. 263–282.

[3] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, pp. 134–160, 2003.

[4] D. Bollegala, N. Noman, and H. Iba, "Rankde: Learning a ranking function for information retrieval using differential evolution," in *Proc. GECCO*. ACM, 2011, pp. 1771–1778.

[5] F. L. Wauthier, M. I. Jordan, and N. Jojic, "Efficient ranking from pairwise comparisons," in *Proc. ICML*, 2013.

[6] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proc. ICGA*, 1987, pp. 224–230.

[7] D. E. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem," in *Proc. ICGA*, 1985, pp. 154–159.

[8] V. A. Cicirello and S. F. Smith, "Modeling GA performance for control parameter optimization," in *Proc. GECCO*, July 2000, pp. 235–242.

[9] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proc. IJCAI*, 1985, pp. 162–164.

[10] V. A. Cicirello, "Non-wrapping order crossover: An order preserving crossover operator that respects absolute position," in *Proc. GECCO 2006*, vol. 2. ACM Press, July 2006, pp. 1125–1131.
[11] L. Hernando, A. Mendiburu, and J. Lozano, "A tunable generator of instances of permutation-based combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. PP, 2015.
[12] M.-H. Tayarani-N and A. Prugel-Bennett, "On the landscape of combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 420–434, 2014.
[13] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Computers & Operations Research*, vol. 34, no. 10, pp. 3143 – 3153, 2007.
[14] V. A. Cicirello and R. Cernera, "Profiling the distance characteristics of mutation operators for permutation-based genetic algorithms," in *Proc. 26th FLAIRS*. AAAI Press, May 2013, pp. 46–51.
[15] K. Sörensen, "Distance measures based on the edit distance for permutation-type representations," *Journal of Heuristics*, vol. 13, no. 1, pp. 35–47, February 2007.
[16] V. A. Cicirello, "On the effects of window-limits on the distance profiles of permutation neighborhood operators," in *Proc. Int. Conf. Bioinspired Information and Communications Technologies*, Dec. 2014, pp. 28–35.
[17] A. Caprara, "Sorting by reversals is difficult," in *Proc. 1st Int. Conf. on Computational Molecular Biology*. ACM, 1997, pp. 75–83.
[18] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.
[19] S. Wright, "Evolution in mendelian populations," *Genetics*, vol. 16, no. 2, pp. 97–159, 1931.
[20] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and ga performance," in *Proc. ECAL*, 1992, pp. 245–254.
[21] M. Mitchell and S. Forrest, "Fitness landscapes: Royal road functions," in *Handbook of Evolutionary Computation*, 1997.
[22] D. Sudholt, "Crossover speeds up building-block assembly," in *Proc. GECCO*. ACM, 2012, pp. 689–702.
[23] G. Reis, F. Fernandéz, and G. Olague, "Cooperative and decomposable approaches on royal road functions: Overcoming the random mutation hill-climber," in *Proc. GECCO*. ACM, 2009, pp. 1875–1876.
[24] A. Fialho, M. Schoenauer, and M. Sebag, "Analysis of adaptive operator selection techniques on the royal road and long k-path problems," in *Proc. GECCO*. ACM, 2009, pp. 779–786.
[25] E. Lutton and J. Levy Vehel, "Holder functions and deception of genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 2, no. 2, pp. 56–71, 1998.
[26] A. M. R. Manso and L. M. P. Correia, "A multiset genetic algorithm for the optimization of deceptive problems," in *Proc. GECCO*. ACM, 2013, pp. 813–820.
[27] M. Li, S. Hill, and C. O'Riordan, "Analysis of a triploid genetic algorithm over deceptive and epistatic landscapes," *ACM SIGAPP Applied Computing Review*, vol. 12, no. 3, pp. 51–59, 2012.
[28] D. E. Goldberg, "Simple genetic algorithms and the minimal, deceptive problem," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed., 1987, pp. 74–88.
[29] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in *Proc. ICGA*. Morgan Kaufmann, 1995, pp. 184–192.
[30] C. Höhn and C. Reeves, "The crossover landscape for the onemax problem," in *Proc. 2nd Nordic Workshop on Genetic Algorithms*, 1996, pp. 27–43.
[31] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, January 1974.
[32] S. Ronald, "More distance functions for order-based encodings," in *Proc. IEEE CEC*. IEEE Press, 1998, pp. 558–563.
[33] M. Sevaux and K. Sörensen, "Permutation distance measures for memetic algorithms with population management," in *Proc. MIC2005*, August 2005, pp. 832–838.
[34] S. Ronald, "Distance functions for order-based encodings," in *Proc. IEEE CEC*. IEEE Press, 1997, pp. 49–54.
[35] S. Ronald, "Finding multiple solutions with an evolutionary algorithm," in *Proc. IEEE CEC*. IEEE Press, 1995, pp. 641–646.
[36] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, June 1938.
[37] M. Meilă and L. Bao, "An exponential model for infinite rankings," *Journal of Machine Learning Research*, vol. 11, pp. 3481–3518, 2010.
[38] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2003.
[39] M. Serpell and J. E. Smith, "Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms," *Evolutionary Computation*, vol. 18, no. 3, pp. 491–514, 2010.
[40] V. A. Cicirello, "On the design of an adaptive simulated annealing algorithm," in *Proc. CP 2007 First Workshop on Autonomous Search*. AAAI Press, September 2007.
[41] C. L. Valenzuela, "A study of permutation operators for minimum span frequency assignment using an order based representation," *Journal of Heuristics*, vol. 7, no. 1, pp. 5–21, 2001.
[42] D. E. Knuth, *The Art of Computer Programming, Volume 1, Fundamental Algorithms*, 3rd ed. Addison Wesley, 1997.
[43] W. P. Swartz, "Automatic layout of analog and digital mixed macro/standard cell integrated circuits," Ph.D. dissertation, Yale University, May 1993.
[44] J. A. Boyan, "Learning evaluation functions for global optimization," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.
[45] J. Lam and J. Delosme, "Performance of a new annealing schedule," in *Proc. 25th ACM/IEEE DAC*, 1988, pp. 306–311.
[46] J. Sarma and K. De Jong, "An analysis of the effects of neighborhood size and shape on local selection algorithms," in *Proc. PPSN IV*. Springer, 1996, pp. 236–244.
[47] S.-Z. Zhao, P. Suganthan, and Q. Zhang, "Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes," *IEEE Trans. Evol. Comput.*, vol. 16, no. 3, pp. 442–446, 2012.
[48] H. Muhlenbein and J. Zimmermann, "Size of neighborhood more important than temperature for stochastic local search," in *Proc. IEEE CEC*, vol. 2, 2000, pp. 1017–1024.
[49] Y.-K. Wang, K.-C. Fan, and J.-T. Horng, "Genetic-based search for error-correcting graph isomorphism," *IEEE Trans. Syst., Man, Cybern. B, Cybern*, vol. 27, no. 4, pp. 588–597, 1997.
[50] V. A. Cicirello and W. C. Regli, "An approach to a feature-based comparison of solid models of machined parts," *AI EDAM*, vol. 16, no. 5, pp. 385–399, 2002.
[51] T. Mantere and J. Koljonen, "Solving, rating and generating sudoku puzzles with ga," in *Proc. IEEE CEC 2007*, 2007, pp. 1382–1389.
[52] Z. Xu, H. Li, and Y. Wang, "An improved genetic algorithm for vehicle routing problem," in *Proc. ICCIS*, 2011, pp. 1132–1135.
[53] D. Romik, *The Surprising Mathematics of Longest Increasing Subsequences*. Cambridge University Press, 2015.

Vincent A. Cicirello (S'95–M'03) was born in Philadelphia, PA, USA, in 1976. He received the B.S. degree in computer science and mathematics from Drexel University, Philadelphia, PA, USA, in 1999, the M.S. degree in computer science from Drexel University, in 1999, and the Ph.D. degree in robotics from Carnegie Mellon University, Pittsburgh, PA, USA, in 2003.

From 2003 to 2005, he was a Research Scientist at the Drexel University Applied Communications and Information Networking Institute, Camden, NJ, USA. He is currently a Professor of Computer Science and Information Systems at Stockton University, Galloway, NJ, USA. His research includes evolutionary computation, computational intelligence, and machine learning.

Dr. Cicirello is a senior member of the Association for Computing Machinery, a life member of the Association for the Advancement of Artificial Intelligence, and a member of the Society for Industrial and Applied Mathematics. He is also a member of ACM SIGAI, ACM SIGCSE, ACM SIGEVO, IEEE Computer Society, and IEEE Computational Intelligence Society.