On the Effects of Window-Limits on the Distance Profiles of Permutation Neighborhood Operators

Vincent A. Cicirello
Richard Stockton College
School of Business
101 Vera King Farris Drive
Galloway, NJ 08205
cicirelv@stockton.edu

ABSTRACT

Local search algorithms, such as simulated annealing, tabu search, and local hill climbers attempt to optimize a solution to a problem by making locally improving modifications to a candidate solution. They rely on a neighborhood operator to restrict the search to a typically small set of possible successor states. The genetic algorithm mutation operator, likewise, enables the exploration of the local neighborhood of candidate solutions within the genetic algorithm's population. In this paper, we profile window-limited variations of several commonly employed neighborhood operators for problems in which candidate solutions are represented as permutations. Window-limited neighborhood operators enable tuning the size of the local neighborhood–e.g., to balance cost of search step against likelihood of getting stuck in a local optima. Window limits potentially can even be adjusted dynamically during search–e.g., to give a stagnated search a "kick." We provide profiles of the distance characteristics of window-limited variations of several of the more common permutation neighborhood operators.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*heuristic methods*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms, permutations and combinations*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling, computations on discrete structures*

General Terms

Algorithms, Theory

Keywords

evolutionary computation, genetic algorithm, local search, mutation, permutation distance, permutation operator, simulated annealing

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257872

1. INTRODUCTION

Local search algorithms, such as simulated annealing, tabu search, and local hill climbers attempt to optimize a solution to a problem by making locally improving modifications to a candidate solution. The mutation operator of genetic algorithms (GA) and other evolutionary computation frameworks, likewise, enables the exploration of the local neighborhood around members of the population of candidate solutions to a problem. One 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 surface is known as the fitness landscape [9], and likely contains a variety of peaks and valleys which represent locally optimal solutions–i.e., solutions for which any small modification leads to a lower quality solution. The objective of local search algorithms is to find an optimal point on that landscape by iteratively exploring the local region of a current candidate solution; or in the case of evolutionary computation algorithms, such as a GA, to explore the local region around populations of solutions. The efficacy of local search, evolutionary computation, or other similar metaheuristics for a given problem is dependent upon its ability to avoid, or escape from, local optima.

Local search algorithms tend to perform better when fitness landscapes are smooth with small numbers of local optima; and are especially challenged when faced with a fitness landscape with plateaus and large numbers of local optima. So it seems that if we had the ability to affect the shape and other characteristics of the fitness landscape for a problem, then we would have some degree of control over the effectiveness of local search for that problem. It turns out that the structure of the fitness landscape depends strongly on how we define the neighborhood of a candidate solution (e.g., [5]). What does it mean for a solution to be "near" another? The shape of the fitness landscape, therefore, depends upon more than the problem alone. It also depends upon the operator used for local improvement. In order to define a relevant fitness landscape that characterizes the problem solving behavior associated with a particular local neighborhood operator, one needs to carefully consider the choice of distance measure used to define what is considered "local."

In this paper, we focus on the permutation representation, in which candidate solutions to an optimization problem are represented as a permutation of the elements of some set. This is a rather natural representation for problems

that involve sequencing—e.g., for the Traveling Salesperson Problem (TSP), a candidate solution is represented as a permutation of the cities. But how should we define the local neighborhood of a permutation? This depends on whether the problem is what has been termed an "A-permutation problem" or an "R-permutation problem" [1, 8], depending on whether absolute or relative positioning is most important to the problem. For example, for the TSP, any circular rotation of a candidate solution is equivalent to any other. Absolute position of a city in the permutation does not particularly matter; rather, it is the relative positions of the cities that are important. For other problems, absolute positioning may have a bigger influence on solution fitness; or perhaps a blend of both properties might be critical.

Our primary objective is to offer guidelines for selecting from among the available local neighborhood operators for a variety of optimization problems. Previously, using several permutation distance measures from the literature, we developed profiles of the most common neighborhood operators that are available for permutations [4]. In this paper, we extend that work and profile window-limited variations of these operators. Window-limited neighborhood operators enable tuning the size of the local neighborhood–e.g., to balance cost of search step against likelihood of getting stuck in a local optima. Window limits potentially can be adjusted dynamically during search–e.g., to give a stagnated search a "kick." A window-limited neighborhood operator restricts the local modification to reside entirely within some smaller "window." Imagine a "swap" operator that randomly swaps the locations of two elements of the permutation. A window-limited swap constrains the indices of the selected elements to be within a specified window of each other. Such window limits on neighborhood operators were first suggested by Lam and Delosme [7] in defining a dynamic simulated annealing schedule to track a theoretical target move acceptance rate by dynamically adjusting the neighborhood size.

The remainder of this paper is organized as follows. In Section 2, we discuss background on permutation distance measures. We present our profiling methodology in Section 3. Then, in Section 4, we profile the distance characteristics of window-limited variations of several of the more common permutation neighborhood operators. We conclude, in Section 5, with a discussion of implications for the design of local search algorithms for permutation problems.

2. BACKGROUND

An important characteristic of fitness landscapes is fitness-distance correlation [6], an application of the Pearson correlation coefficient, which measures the correlation between the fitness of a solution and its distance from the optimal. If fitness (i.e., solution quality) improves the nearer you are to the optimal solution (as distance to the optimal decreases), then searching via a local search algorithm such as simulated annealing or tabu search or via other metaheuristics that rely on searching nearby solutions will be effective.

We therefore require a means of measuring the distance between permutations. Many distance measures for permutations have been proposed [10, 13, 12, 4]. We have chosen distance measures that capture the essence of the so-called "A-Permutation" and "R-Permutation" type problems. In the formalization of distance measures that follow, p_1 and p_2 refer to permutations, $p_1(i)$ refers to the element in position i of the permutation, and N is the permutation's length.

Absolute Position Based Distance Measures: When profiling neighborhood operators, we consider the following distance measures as representative of the characteristics of problems where "absolute" position within the permutation is most critical to the permutation's fitness as a solution:

- **Exact Match:** The exact match distance [10] is an extension of the concept of Hamming distance to permutations. It is the count of the number of positions containing different elements.

$$\delta(p_1, p_2) = \sum_{i=1\ldots N} \begin{cases} 1 & \text{if } p_1(i) \neq p_2(i) \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

- **Deviation Distance:** Deviation distance [10] is the normalized sum of the positional deviations of the elements from one permutation to the other, and is formalized as follows:

$$\delta(p_1, p_2) = \frac{1}{N-1} \sum_{e \in p_1} |i - j|, \text{where } p_1(i) = p_2(j) = e. \quad (2)$$

- **Interchange Distance:** In our earlier work, we introduced a permutation distance called *Interchange Distance*, along with an efficient algorithm for computing it [4]. Interchange Distance is an edit distance measure. The edit distance between two structures (a concept that originated within the string pattern matching community [16]) is the minimum cost of the "edit operations" required to transform one structure into the other. Interchange distance is defined as the minimum number of element by element interchanges (or swaps) needed to transform one permutation into the other—by "swap" we refer to general swaps and not adjacent element exchanges. We previously showed that this distance is an absolute position based distance as it correlates strongly with the Exact Match distance and moderately with Deviation distance, and extremely weakly with distance measures focused on relative position properties [4]. See our prior work for the details of the algorithm we use to compute Interchange Distance.

Relative Position Based Distance Measures: We additionally consider the following distance measure as representative of the characteristics of problems where "relative" position among the elements of the permutation is most critical to the permutation's fitness:

- **R-Type Distance:** The "R-Type" distance [1, 8] was developed specifically to handle permutation problems where relative ordering primarily influences solution fitness. It is the count of the number of adjacent element pairs of p_1 that do not appear as adjacent element pairs in p_2 and can be defined as:

$$\delta(p_1, p_2) = \sum_{i=1}^{N-1} \begin{cases} 0 & \text{if } \exists j, p_1(i) = p_2(j) \text{ and} \\ & \quad p_1(i+1) = p_2(j+1) \\ 1 & \text{otherwise.} \end{cases} \quad (3)$$

Table 1: Average Exact Match distance between original permutation and a random neighbor selected via common neighborhood operators and different window limits. The rows where $L = \infty$ correspond to the basic form of the operator without window limits.

Mutation Operator	Permutation Length 16	32	64	128	256	512	1024
Insertion(1)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Insertion(2)	2.48	2.49	2.50	2.50	2.50	2.50	2.49
Insertion(4)	3.40	3.47	3.48	3.51	3.51	3.49	3.49
Insertion(8)	5.00	5.33	5.39	5.43	5.52	5.46	5.51
Insertion(16)	6.57	8.58	9.11	9.26	9.42	9.48	9.47
Insertion(32)	6.63	12.12	15.76	16.67	17.10	17.38	17.50
Insertion(64)	6.60	12.03	22.76	29.95	32.01	32.58	33.24
Insertion(128)	6.72	11.94	22.50	43.92	58.69	62.28	64.11
Insertion(∞)	6.72	12.06	22.78	44.39	86.15	171.65	344.92
Swap(1)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(2)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(4)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(8)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(16)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(32)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(64)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(128)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Swap(∞)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Scramble(1)	1.00	1.01	0.99	1.00	1.02	1.00	0.99
Scramble(2)	1.48	1.50	1.50	1.48	1.51	1.48	1.50
Scramble(4)	2.41	2.45	2.46	2.49	2.45	2.49	2.52
Scramble(8)	4.03	4.29	4.40	4.46	4.48	4.46	4.49
Scramble(16)	5.69	7.59	8.18	8.29	8.45	8.42	8.46
Scramble(32)	5.70	11.07	14.63	15.74	16.24	16.40	16.28
Scramble(64)	5.71	11.08	21.34	29.30	31.14	32.01	32.05
Scramble(128)	5.64	10.86	21.59	42.55	57.77	61.22	63.01
Scramble(∞)	5.64	10.98	21.74	43.00	85.43	169.99	343.79
Reversal(1)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Reversal(2)	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Reversal(4)	2.90	2.97	2.99	3.00	2.99	2.98	3.00
Reversal(8)	4.56	4.75	4.89	4.96	4.98	4.98	4.97
Reversal(16)	6.15	8.15	8.53	8.83	8.91	9.02	9.01
Reversal(32)	6.24	11.54	14.94	16.19	16.38	16.82	16.64
Reversal(64)	6.25	11.56	22.03	29.52	31.37	32.40	32.82
Reversal(128)	6.24	11.62	21.99	43.74	58.25	62.14	62.97
Reversal(∞)	6.21	11.60	22.22	43.51	85.15	170.81	347.44

Absolute/Relative Hybrid Distance Measures: Some problems are neither strictly an "A-Permutation" problem nor an "R-Permutation" problem. Thus, we consider a distance measure that blends both properties:

- **Reinsertion Distance:** We previously introduced the permutation distance known as *Reinsertion Distance* as the minimum number of removal/reinsertion operations needed to transform one permutation into the other [4]. A removal/reinsertion is an edit operation that removes an element and reinserts it in another location. Reinsertion Distance is a special case of general edit distance for permutations and strings [16, 13], which typically allows three operations (element removal, element insertion, and element relabeling). To compute Reinsertion Distance, we use Wagner and Fischer's dynamic programming algorithm for string edit distance [16] where we assign costs of 0.5 for removals, 0.5 for insertions, and ∞ for replacements.

3. PROFILING METHODOLOGY

3.1 Neighborhood Operators

Our aim is to shed light on when the most common permutation neighborhood operators produce *small* random perturbations (i.e., when they really are selecting states that are nearby the candidate solution). The basic form of the neighborhood operators that we profile are as follows:

- **Insertion** removes one randomly selected element, and reinserts it into a different randomly selected position.
- **Swap** exchanges the positions of 2 randomly selected elements. All other elements remain unchanged.
- **Scramble** selects 2 different random indices, and randomly shuffles the selected region, inclusive of the indices. Scramble can leave the permutation unaltered.
- **Reversal** selects 2 different random indices, and reverses the selected sub-permutation.

Table 2: Average Deviation distance between original permutation and a random neighbor selected via common neighborhood operators and different window limits. The rows where $L = \infty$ correspond to the basic form of the operator without window limits.

Mutation Operator	Permutation Length 16	32	64	128	256	512	1024
Insertion(1)	0.13	0.06	0.03	0.02	0.01	0.00	0.00
Insertion(2)	0.20	0.10	0.05	0.02	0.01	0.01	0.00
Insertion(4)	0.32	0.16	0.08	0.04	0.02	0.01	0.00
Insertion(8)	0.55	0.28	0.14	0.07	0.04	0.02	0.01
Insertion(16)	0.75	0.49	0.26	0.13	0.07	0.03	0.02
Insertion(32)	0.75	0.71	0.47	0.25	0.13	0.06	0.03
Insertion(64)	0.76	0.71	0.68	0.45	0.24	0.12	0.06
Insertion(128)	0.76	0.71	0.69	0.68	0.45	0.24	0.12
Insertion(∞)	0.75	0.71	0.69	0.68	0.67	0.67	0.67
Swap(1)	0.13	0.06	0.03	0.02	0.01	0.00	0.00
Swap(2)	0.20	0.10	0.05	0.02	0.01	0.01	0.00
Swap(4)	0.32	0.16	0.08	0.04	0.02	0.01	0.00
Swap(8)	0.54	0.28	0.14	0.07	0.04	0.02	0.01
Swap(16)	0.76	0.49	0.26	0.13	0.07	0.03	0.02
Swap(32)	0.75	0.71	0.47	0.25	0.13	0.06	0.03
Swap(64)	0.75	0.71	0.69	0.46	0.24	0.13	0.06
Swap(128)	0.76	0.70	0.69	0.68	0.45	0.24	0.12
Swap(∞)	0.75	0.72	0.69	0.68	0.67	0.67	0.68
Scramble(1)	0.07	0.03	0.02	0.01	0.00	0.00	0.00
Scramble(2)	0.12	0.06	0.03	0.01	0.01	0.00	0.00
Scramble(4)	0.26	0.13	0.07	0.03	0.02	0.01	0.00
Scramble(8)	0.65	0.35	0.18	0.09	0.04	0.02	0.01
Scramble(16)	1.26	1.00	0.54	0.28	0.14	0.07	0.04
Scramble(32)	1.27	2.15	1.70	0.94	0.50	0.25	0.13
Scramble(64)	1.24	2.19	3.89	3.22	1.74	0.93	0.47
Scramble(128)	1.28	2.10	3.87	7.44	6.19	3.44	1.75
Scramble(∞)	1.27	2.14	3.85	7.36	14.70	28.60	57.38
Reversal(1)	0.13	0.06	0.03	0.02	0.01	0.00	0.00
Reversal(2)	0.20	0.10	0.05	0.02	0.01	0.01	0.00
Reversal(4)	0.41	0.20	0.10	0.05	0.03	0.01	0.01
Reversal(8)	1.00	0.53	0.27	0.14	0.07	0.03	0.02
Reversal(16)	1.93	1.50	0.82	0.43	0.22	0.11	0.05
Reversal(32)	1.90	3.21	2.59	1.42	0.74	0.38	0.19
Reversal(64)	1.93	3.24	5.84	4.92	2.70	1.40	0.70
Reversal(128)	1.90	3.15	5.85	11.41	9.42	5.16	2.66
Reversal(∞)	1.89	3.23	5.74	11.01	21.35	43.28	86.03

These are the most common permutation neighborhood operators and examples of their usage abound [11, 2, 3, 15].

Window Limited Neighborhood Operators: In this paper, we profile window-limited operators. A window limit, L, on a permutation operator restricts element movement within the permutation by defining the size of a "window" around a selected element within which that element will remain. We define window-limited variations of the above permutation neighborhood operators as follows:

- **Insertion(L)** removes one randomly selected element, and reinserts it into a different randomly selected position no more than L positions away.
- **Swap(L)** exchanges the positions of 2 elements, chosen uniformly at random from among all pairs of elements that are at most L positions apart. All other elements remain in their current positions.
- **Scramble(L)** selects 2 different random indices, and randomly shuffles the selected region, inclusive, with all possible reorderings equally likely. The 2 indices are chosen such that they are at most L elements apart.
- **Reversal(L)** randomly selects 2 different indices, and reverses the selected sub-permutation. The 2 indices are chosen such that they are at most L elements apart.

3.2 Generating Profiling Data

To profile the behavior of the neighborhood operators, we generate 10000 random permutations of each of the following lengths: $N = \{16, 32, 64, 128, 256, 512, 1024\}$. We consider the window limits: $L = \{1, 2, 4, 8, 16, 32, 64, 128\}$. For each combination of N_i, L_j, neighborhood operator, and distance measure, we compute the average distance between the 10000 random permutations and a random neighbor of each. If $L_j >= (N_i - 1)$, then it is equivalent to the basic form with no window limit ($L = \infty$), which we also include.

Table 3: Average Interchange distance between original permutation and a random neighbor selected via common neighborhood operators and different window limits. The rows where $L = \infty$ correspond to the basic form of the operator without window limits.

Mutation Operator	Permutation Length 16	32	64	128	256	512	1024
Insertion(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(2)	1.48	1.49	1.50	1.50	1.50	1.50	1.50
Insertion(4)	2.41	2.46	2.47	2.48	2.48	2.49	2.47
Insertion(8)	4.05	4.30	4.39	4.46	4.46	4.52	4.49
Insertion(16)	5.64	7.62	8.16	8.24	8.39	8.46	8.46
Insertion(32)	5.63	10.97	14.76	16.03	16.13	16.30	16.51
Insertion(64)	5.74	11.08	21.55	28.82	30.88	31.91	31.99
Insertion(128)	5.69	10.95	21.93	43.26	57.24	61.67	63.79
Insertion(∞)	5.67	10.97	21.77	43.01	85.48	172.61	343.67
Swap(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(2)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(4)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(8)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(16)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(32)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(64)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(128)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(∞)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Scramble(1)	0.50	0.51	0.50	0.50	0.49	0.50	0.50
Scramble(2)	0.82	0.82	0.84	0.84	0.84	0.83	0.84
Scramble(4)	1.51	1.56	1.54	1.58	1.57	1.59	1.56
Scramble(8)	2.83	3.07	3.16	3.14	3.22	3.22	3.18
Scramble(16)	4.31	5.89	6.41	6.57	6.66	6.71	6.67
Scramble(32)	4.27	9.12	12.53	13.57	13.89	14.05	14.12
Scramble(64)	4.22	9.15	19.32	26.29	28.27	29.23	29.32
Scramble(128)	4.29	9.11	19.14	39.79	54.12	57.67	59.46
Scramble(∞)	4.27	9.06	19.18	39.31	81.61	168.13	338.66
Reversal(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reversal(2)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reversal(4)	1.46	1.49	1.49	1.50	1.50	1.50	1.49
Reversal(8)	2.29	2.41	2.44	2.48	2.49	2.48	2.50
Reversal(16)	3.07	4.07	4.34	4.40	4.46	4.49	4.52
Reversal(32)	3.16	5.73	7.66	8.07	8.35	8.40	8.49
Reversal(64)	3.11	5.83	10.93	14.63	15.75	16.16	16.45
Reversal(128)	3.11	5.78	10.98	21.62	28.93	30.76	31.77
Reversal(∞)	3.08	5.70	11.21	21.82	43.12	85.60	170.96

4. DISTANCE PROFILES

We begin by profiling the neighborhood operators with respect to the abolute position based distance measures. In our prior work, where window-limited operators were not considered, we showed that the swap operator seemed the ideal absolute position based neighborhood operator, in that regardless of which absolute position based measure we used (Exact Match, Deviation Distance, or Interchange Distance), the swap operator produced a new candidate solution that was near the original–specifically a small constant distance from the original independent of permutation length. We additionally showed that the insertion operator may be appropriate for "A-permutation" problems if the relevant distance measure is Deviation distance; and that for smaller permutation lengths that the reversal operator is appropriate when Interchange Distance is the relevant distance measure. The Scramble operator was always seen as excessively disruptive, as is also generally the case for Reversal.

Here, we now consider the behavior of window-limited variations of these operators. The results are summarized in Tables 1, 2, and 3. First, consider the Exact Match distance (Table 1) and the Interchange Distance (Table 3). The swap operator, independent of the window limit and permutation length, produces a new candidate solution that is an Exact Match distance of 2 from the original and an Interchange Distance of 1 from the original. Without window limits, all of the other operators are rather disruptive with respect to these distances. However, with window limits, the average distance between the original and modified permutations can be kept as small as desired if the insertion, scramble, or reversal operators are used, even for large permutations. Deviation Distance (see Table 2) is the trickier case. When window-limits are introduced, the deviation distance between offspring permutations and their parents diminishes toward 0 as permutation length increases regardless of which neighborhood operator is used to produce the offspring. Without window-limits, both the swap and in-

Table 4: Average R-Type distance between original permutation and a random neighbor selected via common neighborhood operators and different window limits. The rows where $L = \infty$ correspond to the basic form of the operator without window limits.

Mutation Operator	Permutation Length 16	32	64	128	256	512	1024
Insertion(1)	2.86	2.94	2.97	2.98	2.99	3.00	3.00
Insertion(2)	2.87	2.94	2.97	2.99	2.99	3.00	3.00
Insertion(4)	2.86	2.93	2.97	2.99	2.99	3.00	3.00
Insertion(8)	2.82	2.93	2.97	2.98	2.99	3.00	3.00
Insertion(16)	2.76	2.92	2.96	2.98	2.99	3.00	3.00
Insertion(32)	2.75	2.87	2.96	2.98	2.99	3.00	3.00
Insertion(64)	2.76	2.87	2.93	2.98	2.99	3.00	3.00
Insertion(128)	2.75	2.88	2.94	2.97	2.99	3.00	3.00
Insertion(∞)	2.75	2.87	2.94	2.97	2.98	2.99	3.00
Swap(1)	2.87	2.94	2.97	2.98	2.99	3.00	3.00
Swap(2)	3.35	3.43	3.46	3.49	3.49	3.49	3.49
Swap(4)	3.58	3.68	3.71	3.73	3.74	3.75	3.75
Swap(8)	3.66	3.79	3.83	3.85	3.87	3.87	3.88
Swap(16)	3.63	3.83	3.90	3.92	3.93	3.93	3.94
Swap(32)	3.62	3.81	3.92	3.95	3.96	3.96	3.97
Swap(64)	3.63	3.81	3.90	3.96	3.98	3.98	3.98
Swap(128)	3.63	3.81	3.91	3.96	3.98	3.99	3.99
Swap(∞)	3.62	3.81	3.91	3.95	3.98	3.99	3.99
Scramble(1)	1.46	1.48	1.48	1.49	1.49	1.50	1.50
Scramble(2)	1.98	2.04	2.04	2.07	2.07	2.08	2.07
Scramble(4)	2.98	3.07	3.15	3.14	3.16	3.17	3.17
Scramble(8)	4.58	5.02	5.13	5.22	5.22	5.24	5.29
Scramble(16)	6.18	8.43	9.02	9.13	9.29	9.33	9.34
Scramble(32)	6.31	11.68	15.67	16.60	16.97	17.10	17.27
Scramble(64)	6.30	11.68	22.70	29.83	32.19	32.85	33.23
Scramble(128)	6.20	11.75	22.53	43.67	58.01	62.24	63.34
Scramble(∞)	6.30	11.79	22.46	44.45	87.12	172.27	345.46
Reversal(1)	2.87	2.93	2.97	2.98	2.99	3.00	3.00
Reversal(2)	3.35	3.43	3.46	3.49	3.49	3.49	3.50
Reversal(4)	4.26	4.39	4.42	4.48	4.48	4.51	4.50
Reversal(8)	5.89	6.23	6.40	6.46	6.52	6.49	6.48
Reversal(16)	7.39	9.47	10.03	10.34	10.32	10.50	10.44
Reversal(32)	7.39	12.91	16.72	17.57	18.23	18.39	18.35
Reversal(64)	7.35	12.93	23.75	31.09	32.60	33.70	34.20
Reversal(128)	7.43	12.92	23.67	45.29	59.07	62.85	65.95
Reversal(∞)	7.43	12.91	23.61	44.95	88.85	171.95	348.49

sertion operators produce relatively small modifications of approximately constant magnitude. For fitness landscapes best characterized by Deviation Distance, one must be careful in the setting of the window limit to ensure that the neighborhood considered is not too small.

Next, we profile the neighborhood operators with respect to the relative position based distance measure, R-Type distance. The results are shown in Table 4. In our prior work, we showed that both the swap and insertion operators produce offspring that are a small constant average distance from the parent permutations independent of permutation length (R-Type distance of around 3 for insertion and around 4 for swap). Here, those results are reaffirmed, as well as strengthened. In the limit as the permutation length increases, the R-Type distance between an original permutation and a random neighbor generated with **Swap**(L) rapidly approaches 4, independent of the window size L. Likewise, in the limit as the permutation length increases, the R-Type distance between an original permutation and a random neighbor generated with **Insertion**(L) rapidly approaches 3. Additionally, although previously we showed both scramble and reversal to be too disruptive without window limits, here we find that window limits can be used to control the magnitude of the modifications made by scramble and reversal to our desired level.

Finally, we profile the neighborhood operators for the Reinsertion Distance to characterize their behavior for fitness landscapes for permutation problems for which a blend of the absolute positions and relative positions of the elements have an influence over solution fitness. The results can be found in Table 5. The insertion neighborhood operator produces offspring a constant reinsertion distance of 1 from the parent, independent of permutation length and independent of the window limit—this follows by the definition of the reinsertion distance measure itself. Similarly, the swap operator produces offspring permutations that are an average

Table 5: Average Reinsertion distance between original permutation and a random neighbor selected via common neighborhood operators and different window limits. The rows where $L = \infty$ correspond to the basic form of the operator without window limits.

Mutation Operator	Permutation Length 16	32	64	128	256	512	1024
Insertion(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(2)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(4)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(8)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(16)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(32)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(64)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(128)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Insertion(∞)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Swap(2)	1.49	1.49	1.51	1.50	1.51	1.50	1.50
Swap(4)	1.72	1.73	1.75	1.75	1.75	1.74	1.75
Swap(8)	1.83	1.86	1.87	1.87	1.87	1.87	1.88
Swap(16)	1.87	1.92	1.93	1.93	1.94	1.94	1.94
Swap(32)	1.88	1.94	1.96	1.96	1.96	1.97	1.97
Swap(64)	1.88	1.93	1.97	1.98	1.98	1.98	1.98
Swap(128)	1.87	1.94	1.97	1.98	1.99	1.99	1.99
Swap(∞)	1.88	1.94	1.97	1.98	1.99	2.00	2.00
Scramble(1)	0.50	0.50	0.50	0.50	0.50	0.50	0.50
Scramble(2)	0.73	0.75	0.75	0.75	0.74	0.75	0.76
Scramble(4)	1.26	1.30	1.32	1.32	1.31	1.33	1.33
Scramble(8)	2.29	2.50	2.58	2.57	2.60	2.61	2.63
Scramble(16)	3.44	4.76	5.15	5.38	5.40	5.41	5.47
Scramble(32)	3.47	7.46	10.26	11.12	11.38	11.57	11.64
Scramble(64)	3.44	7.40	15.95	21.95	23.72	24.46	24.74
Scramble(128)	3.46	7.31	15.96	34.38	46.99	50.74	52.07
Scramble(∞)	3.47	7.46	16.05	34.32	72.43	148.83	312.71
Reversal(1)	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Reversal(2)	1.48	1.49	1.50	1.50	1.50	1.50	1.50
Reversal(4)	2.40	2.47	2.48	2.48	2.49	2.50	2.52
Reversal(8)	4.07	4.30	4.41	4.46	4.49	4.50	4.51
Reversal(16)	5.64	7.56	8.14	8.28	8.38	8.43	8.40
Reversal(32)	5.64	10.94	14.87	15.83	16.15	16.34	16.31
Reversal(64)	5.70	10.92	21.73	29.07	31.23	31.87	32.60
Reversal(128)	5.64	10.96	21.54	42.81	57.76	62.13	62.72
Reversal(∞)	5.61	11.11	21.84	43.16	85.77	171.02	343.24

distance from the parent permutations that is approaching 2 as permutation length increases, independent of the window limit. Previously, without window limits, we found both scramble and reversal to be too disruptive. However, as is the case with the other distance measures, we can use window limits to fine-tune the magnitude of the modifications they make to the current candidate solution in our search.

5. CONCLUSIONS

In this paper, we explored the effects of window-limits on the distance characteristics of several commonly employed local search operators for the permutation representation. The results are consistent with our prior profiling of the basic form of the neighborhood operators, and confirm the status of certain operators as especially relevant for either "A-permutation" or "R-permutation" problems. However, with window limits, operators that are otherwise too disruptive, are good candidates for algorithms in which it is desirable to be able to tune the size of the local neighborhood.

With respect to "A-permutation" problems, we previously saw that the swap operator and, depending upon the specific absolute position based distance used, the insertion operator consistently make small local modifications to candidate solutions. And thus, seem particularly well-suited to "A-permutation" problems. We also previously saw that for "R-permutation" problems, and for problems that cannot be easily categorized as either strictly "A-permutation" or "R-permutation" problems, that both the swap and insertion operator are solid choices and will move the search to "nearby" solutions on fitness landscapes characterized by the relative position based distance measures.

Now that we have examined the effects of window limits, these results have been reaffirmed. However, we have additionally discovered behavior of the other neighborhood operators that is particularly interesting, and which lend some guidance to the design of local search algorithms for permutation problems. First, although otherwise overly dis-

ruptive, both the scramble and reversal operators can be controlled to some extent via window limits. With any fixed window limit, these operators produce offspring that are approximately a constant average distance from the parent permutations for all of the distance measures considered except for deviation distance, independent of permutation length. The average distances in these cases are proportional to the window limit. Thus, we can easily increase or decrease the size of the local neighborhood by increasing or decreasing the window limit used with these operators. This makes the scramble and reversal permutation operators particularly suitable for local search algorithms that dynamically adjust the size of the local neighborhood, such as simulated annealing using either Lam and Delosme's annealing schedule [7] or using the more practical Modified Lam Schedule [14]. The insertion operator, when used for "A-permutation" problems, is likewise well-suited to this sort of neighborhood size tuning. The window limits have no effect on the distance characteristics of swap, and no effect on the distance characteristics of the insertion operator for "R-permutation" problems or blended problems.

6. REFERENCES

[1] V. Campos, M. Laguna, and R. Martí. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing*, 17(1):111–122, 2005.

[2] V. A. Cicirello. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06)*, pages 1125–1131. ACM Press, July 2006.

[3] V. A. Cicirello. On the design of an adaptive simulated annealing algorithm. In *Proceedings of the ICAPS First Workshop on Autonomous Search*. AAAI Press, September 2007.

[4] V. A. Cicirello and R. Cernera. Profiling the distance characteristics of mutation operators for permutation-based genetic algorithms. In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*, pages 46–51. AAAI Press, May 2013.

[5] J. Czogalla and A. Fink. Fitness landscape analysis for the resource constrained project scheduling problem. In T. Stützle, editor, *Learning and Intelligent Optimization*, pages 104–118. Springer-Verlag, 2009.

[6] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. 1995.

[7] J. Lam and J. Delosme. Performance of a new annealing schedule. In *Proc of the 25th ACM/IEEE Design Automation Conference*, pages 306–311, 1988.

[8] R. Martí, M. Laguna, and V. Campos. Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems. In *Metaheuristic Optimization via Memory and Evolution*, pages 263–282. Springer, 2005.

[9] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[10] S. Ronald. More distance functions for order-based encodings. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 558–563. IEEE Press, 1998.

[11] M. Serpell and J. E. Smith. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, 18(3):491–514, 2010.

[12] D. Shapira and J. A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380–392, June 2007.

[13] K. Sörensen. Distance measures based on the edit distance for permutation-type representations. *Journal of Heuristics*, 13(1):35–47, February 2007.

[14] W. P. Swartz. *Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integrated Circuits*. PhD thesis, Yale University, May 1993.

[15] C. L. Valenzuela. A study of permutation operators for minimum span frequency assignment using an order based representation. *Journal of Heuristics*, 7(1):5–21, 2001.

[16] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, January 1974.