Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position

Vincent A. Cicirello
The Richard Stockton College of New Jersey
Computer Science and Information Systems
P.O. Box 195
Pomona, NJ 08240
cicirelv@stockton.edu

ABSTRACT

In this paper, we introduce a new crossover operator for the permutation representation of a GA. This new operator—Non-Wrapping Order Crossover (NWOX)—is a variation of the well-known Order Crossover (OX) operator. It strongly preserves relative order, as does the original OX, but also respects the absolute positions within the parent permutations. This crossover operator is experimentally compared to several other permutation crossover operators on an NP-Hard problem known as weighted tardiness scheduling with sequence-dependent setups. A GA using this NWOX operator finds new best known solutions for several benchmark problem instances and proves to be superior to the previous best performing metaheuristic for the problem.

Categories and Subject Descriptors: I.2.8 [Problem Solving, Control Methods, Search]: Heuristic Methods

General Terms: Algorithms

Keywords: permutation representation, order-based crossover, position-based crossover, weighted tardiness scheduling, genetic algorithms

1. INTRODUCTION

There are many problems that are well-suited to the application of a genetic algorithm (GA), but for which the standard bit-string representation poses challenges. Sequencing problems are one class of such problems (e.g., the traveling salesperson (TSP), and some scheduling problems). Although a GA using the bit-string representation can be quite successful for these problems, it is not the most natural problem encoding. One alternative that many have turned to over the years is the permutation representation. Rather than encoding the problem as a bit-string, one uses a permutation of the N elements (e.g., the cities of a TSP or the jobs of the scheduling problem) that must be sequenced.

Specialized recombination operators are necessary to handle the intricacies of the permutation representation. Over the years, many crossover operators for the permutation representation have been designed (e.g., [13, 7, 6, 4, 17, 11, 15, 8]). Some of these are largely problem dependent such as the edge assembly crossover (EAX) operator for the TSP [17]. Others can be generally classified into a couple broad categories: (a) position-based crossover, (b) order-based crossover, and (c) hybrid crossover operators. *Position-based* crossover operators (e.g., [13]) tend to respect the absolute position of the genetic material of the parents during recombination; while the *order-based* crossover operators (e.g., [6]) tend to respect the relative position of the genetic material of the parents. For example, if you consider the traveling salesperson problem for illustrative purposes, a position-based crossover operator would tend to preserve the absolute position of the cities from the parent tours when constructing the children; while an order-based crossover operator would tend to preserve the relative position of the cities from the parents when constructing the children. Other operators tend to lie somewhere in the middle between position-based and order-based (e.g., [7, 4]).

In this paper, we present a novel variation of the well-known Order Crossover (OX) [6]. OX is strongly order-based. Our new variation that we call Non-Wrapping Order Crossover (NWOX) is designed to retain the strong order-based characteristic of OX, but also to respect the absolute positioning of the values within the parent permutations. Our motivation is the Weighted Tardiness Scheduling Problem with Sequence-Dependent Setups. This NP-Hard scheduling problem at the present time limits complete algorithms that guarantee optimal solutions to solving instances no larger than approximately 20-30 jobs [14]. Heuristic or metaheuristic algorithms are thus a necessity for problem instances significantly larger. This is a problem for which the order of jobs in the sequence is highly important to the fitness of the solutions, but for which absolute position is also important to an extent. The GA that we design using our NWOX operator is able to effectively solve instances of this problem, including finding new best known solutions to several difficult benchmark instances, and in general outperforms the previous best metaheuristic for the problem.

The paper will proceed as follows. We begin in Section 2 by overviewing several existing permutation crossover operators. Next, in Section 3 we formalize the weighted tardiness scheduling problem. We then present the NWOX operator in Section 4. This is followed by experimental results in Section 5 and our paper concludes in Section 6.

2. CROSSOVER OPERATORS FOR A PERMUTATION REPRESENTATION

In the study presented in this paper, five permutation crossover operators are considered. One of these is position-based (CX) and has existed in the literature for some time. A second is order-based (OX) and is also well known. Three more fall somewhere in the middle. Two, PMX and UPMX, are fairly well-known. The last is a novel operator that brings an element of position preservation to an existing order-based operator and is introduced in Section 4.

Cycle Crossover (CX). CX [13] begins by initializing the value of the first position of child C_1 with the value $v_1(1)$ in the first location in parent P_1. It then looks at the value $v_2(1)$ of the first location of parent P_2. It searches P_1 for $v_2(1)$. Consider that it is found in position i. Position i of child C_1 is then given the value $v_2(1)$. This procedure is then repeated for the value $v_2(i)$ that is located in the i-th position of parent P_2 and so forth until a cycle is completed (i.e., until the next $v_2(i)$ is already in child C_1). At this point, all positions j in C_1 that have not yet been given a value are assigned the value $v_2(j)$ that is in the j-th position of parent P_2. This procedure is then repeated to generate child C_2 in a similar manner.

Partially Matched Crossover (PMX). PMX [7] begins by initializing two children permutations C_1, C_2 with copies of the parents P_1, P_2. Two positions, a and b, are then selected uniformly at random from the interval $[1, L]$, where L is the length of the permutations. Without loss of generality, assume $a \leq b$. Let $v_1(a)$, $v_2(a)$ be the values of the a-th position of parents P_1 and P_2 respectively. The locations of $v_1(a)$, $v_2(a)$ in child C_1 are determined and then swapped; likewise for child C_2. This swapping procedure is then repeated using the values $v_1(a+1)$, $v_2(a+1)$ and so forth up to and including the pair $v_1(b)$, $v_2(b)$.

Uniform Partially Matched Crossover (UPMX). Using the same swapping procedure as PMX is UPMX [4]. Unlike PMX, UPMX does not select a cross region to dictate which swaps to make. Instead, UPMX separately considers each position $i \in \{1 \ldots L\}$. With probability U, the values $v_1(i)$, $v_2(i)$ are used by the same swapping procedure as PMX. For the experiments of this paper, $U = 0.33$ which results in the same number of swaps per cross on average as PMX.

Order Crossover (OX). OX [6] begins by initializing two children C_1, C_2 with copies of the parents P_1, P_2 (see Figure 1(a)). Two positions, a and b, are then selected uniformly at random from the interval $[1, L]$ (see Figure 1). Again assume $a \leq b$. Child C_1 is searched for the locations of values $v_2(a), v_2(a+1), \ldots, v_2(b)$. Those locations are replaced by "holes" (see Figure 1(b)). This procedure is repeated inserting "holes" in C_2 in the place of values $v_1(a), v_1(a+1), \ldots, v_1(b)$. Next a "sliding" motion is used to move the "holes" into the $a \ldots b$ region of each child (see Figure 1(c)). That is, all non-holes are slid leftward until they are grouped together in one contiguous string (keeping their original order). This group is then further slid leftward with the values in the leftmost positions wrapping to the right end of the child (if necessary) until there are no non-hole values in positions $a \ldots b$. Values $v_2(a), v_2(a+1), \ldots, v_2(b)$ of parent P_2 are then placed in positions $a \ldots b$ of child C_1; and likewise for values $v_1(a), v_1(a+1), \ldots, v_1(b)$ of parent P_1 in child C_2 (see Figure 1(d)).

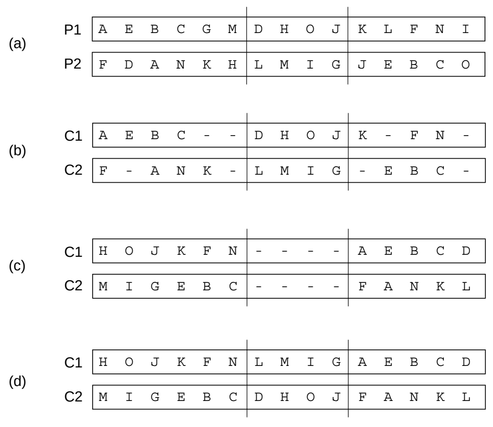

(a)
P1 A E B C G M D H O J K L F N I
P2 F D A N K H L M I G J E B C O
(b)
C1 A E B C - - D H O J K - F N -
C2 F - A N K - L M I G - E B C -
(c)
C1 H O J K F N - - - - A E B C D
C2 M I G E B C - - - - F A N K L
(d)
C1 H O J K F N L M I G A E B C D
C2 M I G E B C D H O J F A N K L

Figure 1: Example of the steps taken by OX

3. MINIMIZING WEIGHTED TARDINESS W/ SEQUENCE DEPENDENT SETUPS

The problem of Weighted Tardiness Scheduling with Sequence-Dependent Setups is encountered in a number of real-world applications (e.g., turbine component manufacturing [3], the packaging industry [1], among others [12]). In this problem we are given a set of jobs $J = \{j_0, j_1, \ldots, j_N\}$. Each of the jobs j has a weight w_j, duedate d_j, and process time p_j. Furthermore, $s_{i,j}$ is defined as the amount of setup time required prior to the start of job j if it is to follow job i on the machine. It is not necessarily the case that $s_{i,j} = s_{j,i}$. The 0-th "job" simply indicates the starting point of the problem ($p_0 = 0$, $d_0 = 0$, $s_{i,0} = 0$, $w_0 = 0$). Its purpose is for the specification of the setup time of each of the jobs if sequenced in position 1. The sequence-dependent nature of the setup times is a primary source of problem difficulty. The particular version of the problem that we concern ourselves with here is the case where the N jobs must be sequenced on a single machine and where preemption of a job during setup or processing is not permitted. Furthermore, if a machine is processing or setting up, then it cannot do anything else until that operation is complete.

The objective of this problem is to sequence the set of jobs J on a machine to minimize the total weighted tardiness:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(c_j - d_j, 0), \tag{1}$$

where T_j is the tardiness of job j. The completion time c_j of job j is equal to the sum of the process and setup times of all jobs that come before it in the sequence plus the setup and process time of job j itself. Specifically, let $\pi(j)$ be the position in the sequence of job j. We can now define c_j as:

$$c_j = \sum_{i,k \in J, \pi(i) <= \pi(j), \pi(i) = \pi(k)+1} p_i + s_{k,i}. \tag{2}$$

The weighted tardiness scheduling problem with sequence-dependent setups is NP-hard in the strong sense.

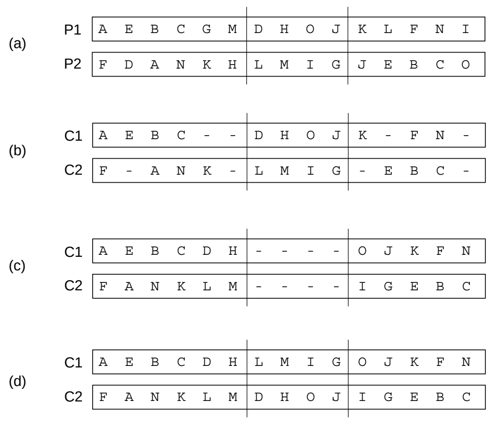
4. NON-WRAPPING ORDER CROSSOVER

For some problems represented as permutations, the key building blocks of a solution derive from the absolute position in the permutation. For others, the relative positions are most important. However, for some problems such as that of weighted tardiness scheduling with sequence-dependent setups (described in Section 3), high quality solutions derive from both the absolute position of the jobs within the permutation as well as the relative ordering of those jobs. For example, absolute position is important since jobs with very high weights and early duedates probably need to occur towards the beginning of the sequence; while jobs with low weights and loose duedates might be able to go towards the end of the sequence. At the same time, relative position of the jobs is also important due to the sequence-dependent nature of the setup times. With this in mind we set out to design a GA for the problem. In considering the use of a permutation representation, we of course had to consider the available crossover operators.

Position-based operators such as CX are good at deriving child schedules that respect the absolute positions of the jobs in the parent schedules. However, in many cases the child schedules can lose sub-sequences of the parents that are vital to their fitness. Consider an example where we have two parents: $P_1 = \{a, e, b, c, \ldots\}$ and $P_2 = \{f, d, a, n, \ldots\}$ (only the beginning of these permutations are shown). Now consider that jobs e and b in parent P_1 have tight duedates and need to be scheduled relatively early in the sequence as they are in P_1. However, also consider that the setup time for b if it follows e is very low, but that if it followed any other job would be much higher. In other words, it is not only important for b to be early in the sequence, but it is also important for it to follow e. A crossover operator that is focused solely on preserving position information will miss this key ordering information. For example, one possible pair of children may look like: $C_1 = \{a, d, b, c, \ldots\}$ and $C_2 = \{f, e, a, n, \ldots\}$. If the setup cost for job b following job d is very high compared to following job e, then although job b is still relatively early in the sequence it may be processed much later in time. This can have a chain reaction effect pushing many other jobs later in time.

Contrast this with crossover operators that focus on preserving relative order such as OX. OX handles this situation nicely in that jobs e and b will likely carry over into one of the children permutations retaining their relative positions. However, as specified, the "sliding" motion of the OX operator has a tendency of wrapping values from the beginning of the parent permutations to the end of the child permutations. So although jobs e and b may retain their relative positions, there is a chance they may do so at the end of the children. If the scheduling problem is even of a small size, this can have drastic effects on the fitness of the children.

Noting these features of the existing crossover operators for permutations, we now present a variation of OX that we call Non-Wrapping Order Crossover (NWOX). It uses a modified version of the sliding action of the original OX. NWOX begins by initializing two children C_1, C_2 with copies of the parents P_1, P_2. Two positions, a and b, are then selected uniformly at random from the interval $[1, L]$ (see Figure 2(a)). Assume $a \leq b$. Child C_1 is searched for the locations of values $v_2(a), v_2(a+1), \ldots, v_2(b)$. Those locations are replaced by "holes" (see Figure 2(b)). This procedure is repeated inserting "holes" in C_2 in the place of values $v_1(a), v_1(a+1), \ldots, v_1(b)$. Next a "sliding" motion is used to move the "holes" into the $a \ldots b$ region of each child. That is, all non-holes are slid leftward until they are grouped together in one contiguous string (keeping their original order). Here is where NWOX differs from OX. Rather than continuing to slide the non-holes leftward wrapping to the right end as OX does, NWOX instead slides any non-holes that appear in positions $a \ldots b$ to the right while leaving "holes" in that region (see Figure 2(c)). NWOX then concludes as in OX by placing values $v_2(a), v_2(a+1), \ldots, v_2(b)$ of parent P_2 in positions $a \ldots b$ of child C_1; and likewise for values $v_1(a), v_1(a+1), \ldots, v_1(b)$ of parent P_1 in child C_2 (see Figure 2(d)). If you compare the children in the example of Figure 2(d) to their parents in Figure 2(a), you can note that although the sliding motion moves almost every allele value to a new location, none of the jobs move very far from their original positions in the parents. Particularly note that no jobs on the left end of the parents wrap to the right end of the children. Relative order of the jobs is transmitted from parents to children to the same degree as in the original OX.



(a)
P1 A E B C G M D H O J K L F N I
P2 F D A N K H L M I G J E B C O
(b)
C1 A E B C - - D H O J K - F N -
C2 F - A N K - L M I G - E B C -
(c)
C1 A E B C D H - - - - O J K F N
C2 F A N K L M - - - - I G E B C
(d)
C1 A E B C D H L M I G O J K F N
C2 F A N K L M D H O J I G E B C

Figure 2: Example of the steps taken by NWOX

5. RESULTS

In this section we detail experiments that we conducted involving the weighted tardiness scheduling problem with sequence-dependent setups that was formalized in Section 3. Specifically, we employ the set of benchmark instances that were generated using a procedure proposed by Lee et al [10] and used by Cicirello and Smith [5] in a study of various metaheuristic algorithms for the problem. There are 120 problem instances, each with 60 jobs. The performance of many algorithms for this problem is discussed in [5], including the current best metaheuristic algorithm.

Unless otherwise stated, in all of the following experiments, we use a population size of 100 and an elitism model that carries the best individual into the next generation unaltered. Stochastic Universal Sampling [2] is then used to select the rest of the next generation. Since our objective function needs to be minimized, we cannot directly employ the objective function for the fitness function. Instead, we define fitness of an individual i as:

$$F_i = 1 + \max_{j=1}^{100}\{T_j\} - T_i, \tag{3}$$

where T_i is the total weighted tardiness of the schedule given by permutation i. All but the elitist individual is subject to possible crossover and mutation according to the crossover and mutation rates. The crossover and mutation rates for each experiment will be discussed in the appropriate subsection below, as will the mutation operator.

5.1 Comparing the Crossover Operators

In the first set of experiments, we set out to compare the NWOX operator to other crossover operators. Specifically, we compare NWOX to OX, CX, PMX, and UPMX. In this first set of experiments, no mutation operator is used to ensure that we can independently examine the effects of the various crossover operators. The crossover rate is set to 1.0. Individuals are randomly paired for recombination.

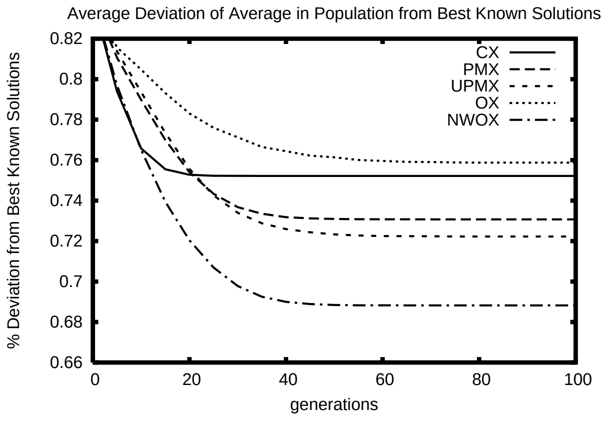
The results are averages of 10 runs for each of the 120 problem instances from the benchmark set (a total of 1200 runs) and can be seen in Figure 3. In Figure 3(a) is shown the percentage deviation of the average individual in the population from the best known solutions as it varies with time (for 100 generations of the algorithm). Figure 3(b) likewise shows the deviation from the best known solutions of the best individual in the population with time. Both graphs show averages of 1200 runs (10 per problem instance).

Lacking mutation to prevent search stagnation, the search converges upon a population composed entirely of copies of a single individual in no more than 40-50 generations independent of which crossover operator is used. Note the leveling off of all of the curves by around generation 40-50. Interestingly, the strongly position-based operator CX and the strongly order-based operator OX perform similarly. CX does however dominate OX for short searches of 40 or less generations. The next noteworthy result is that the position-based/order-based hybrid operators of PMX and UPMX do a much better job of recombining the genetic material of the parent chromosomes than do either CX or OX. This should be expected from the nature of the problem as discussed in Sections 3 and 4 in that for high fitness individuals both position and order information is important.

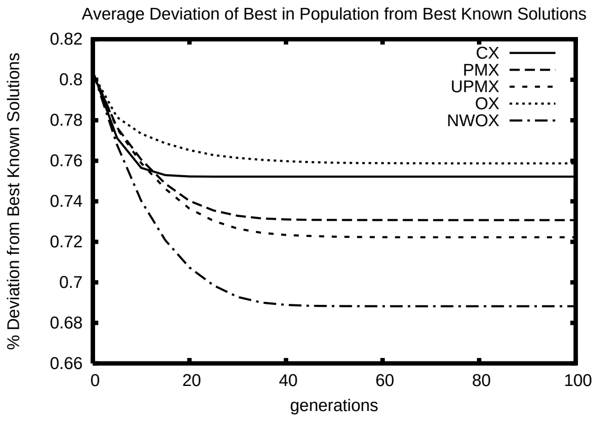
The most important point here, however, is that the new NWOX operator clearly dominates the other four in this study. For the first 10-15 generations, NWOX and CX have equivalent performance. At that point, however, CX's performance completely levels off (the search stagnates). NWOX's performance continues to improve (deviation from best known solutions decreases) dramatically through the first 40 generations dominating the other four operators. From these results, we hypothesize that during the early phases of the search (first 10-15 generations), the absolute positioning of the jobs is most important since NWOX's performance very closely tracks that of the strongly position-based CX. After that initial phase, the order-preserving power of NWOX kicks in. Once the population converges upon "good" general positioning (e.g., jobs that need to be processed earlier towards the beginning of the sequence), the importance of fine-tuning the relative order of the jobs is highlighted by NWOX's continued improvement and CX's complete stagnation.

5.2 Effects of Mutation

If we are to design a GA with a permutation representation and the NWOX operator, it becomes necessary to introduce a mutation operator to prevent the search from



(a)

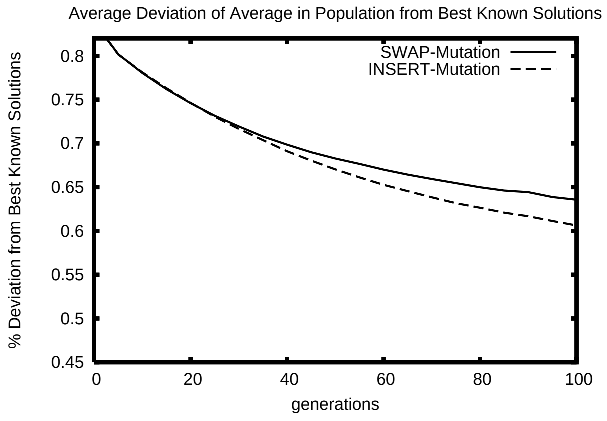


(b)

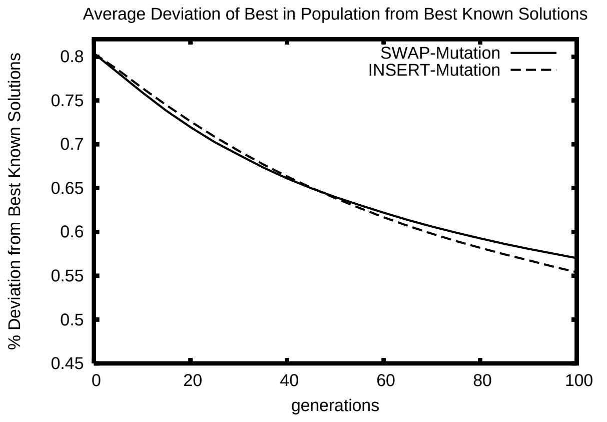
Figure 3: Comparison of the crossover operators: (a) percentage deviation of average in population from best known solutions; (b) percentage deviation of best in population from best known solutions.

stagnating. We consider here two alternatives that we call: (a) Swap Mutation; and (b) Insert Mutation. The *Swap* operator randomly selects two positions in the permutation and swaps the jobs located in those positions. The *Insert* operator removes a random job from the permutation and re-inserts it into a randomly selected position. These were the local hill-climbing operators used previously by Lee et al [10] as part of their algorithm. Figure 4 shows the results of comparing these two mutation operators. No crossover operator was used in this set of experiments to ensure that the effects we are comparing are truly due to the mutation operators alone. An elitism model is again used retaining the best individual of the population unaltered, as is Stochastic Universal Sampling. The mutation rate is set to 1.0 meaning that all individuals in the new population undergo mutation. That is, if swap mutation is used then each individual has exactly one swap, and similarly for the insert mutator.

Figure 4 shows percentage deviation of the average population fitness from the best known solutions (Figure 4(a)) and the same for the best individual in the population (Figure 4(b)). These are averages of 10 runs across 120 problem



(a)



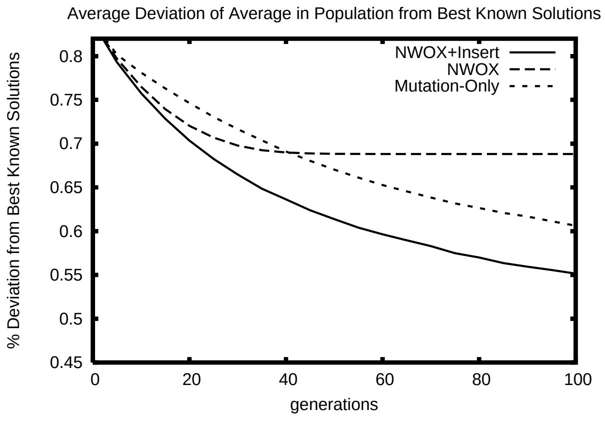
(b)

Figure 4: Comparison of the mutation operators: (a) percentage deviation of average in population from best known solutions; (b) percentage deviation of best in population from best known solutions.

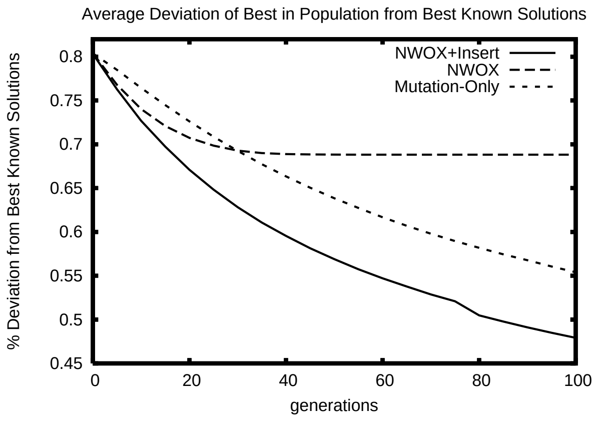
instances (total of 1200 runs). The two mutation operators perform similarly, with the insert operator having a slight edge on the swap operator.

5.3 The Tuned Algorithm

In this third set of experiments, we set out to design a GA that uses both the NWOX operator and mutation to prevent the search from early convergence. We chose the better of the two mutation operators—Insert mutation. We then began by generating a small set of problem instances separate from the benchmark set to use for algorithm tuning. Using a small set of 30 problem instances (not contained in the benchmark set), we applied a simple automated parameter tuning algorithm that systematically tweaked the crossover, mutation, and elitism rates. We held the population size constant at 100. The end result was a GA with an elitism model that retains the best 3 individuals of the population into the next generation unaltered. The crossover rate is 0.95 (random pairings) using NWOX and the mutation rate is 0.65 using Insert mutation. A single Insert mutation is applied to any individual chosen for mutation.



(a)



(b)

Figure 5: Comparison of: (1) a GA using NWOX and the Insert mutator; (2) a GA using NWOX alone; and (3) a GA using Insertion mutation alone. Plots show: (a) percentage deviation of average in population from best known solutions; (b) percentage deviation of best in population from best known solutions.

Figure 5 shows the results of comparing the tuned algorithm (NWOX+Insert) to crossover alone and to mutation alone.

The comparison of Figure 5 again shows the percentage deviation of the average individual (Figure 5(a)) and best individual (Figure 5(b)) of the population from the best known solutions averaged over 10 runs across the 120 problem instances of the benchmark set (a total of 1200 runs). The first noteworthy point is that NWOX (without a mutator) dominates mutation alone only until the point where the population converges (around generation 40), where mutation alone takes the lead over crossover alone. When you add a mutation operator to the GA, we find that the combination of NWOX and Insert mutation dominates the use of either operator when used independently of each other. The Insert mutation operator is effective at preventing the search from stagnating, while the NWOX crossover operator effectively recombines the genetic material of the parent population into new and improved schedules.

Table 1: Comparison of NWOX-Insert with current best metaheuristics for the weighted tardiness scheduling problem with sequence-dependent setups. %Δ is the average percent deviation from the current best known solutions. Also shown is the number of solutions evaluated by the algorithm as well as the number of search nodes generated.

Algorithm	%Δ	Evaluated	Generated
NWOX-Insert (10k gens)	9.9%	953,125	1,552,100
VBSS-HC (10k restarts)	12.2%	20,000	18,818,609
NWOX-Insert (5k gens)	12.3%	476,613	776,100
VBSS-HC (5k restarts)	13.4%	10,000	9,412,105
LDS	16.2%	1,533,116	912,864,816
NWOX-Insert (1k gens)	17.0%	95,403	155,300
VBSS-HC (1k restarts)	17.7%	2,000	1,883,579
DDS (depth 3)	22.6%	205,320	339,393,960
NWOX-Insert (500 gens)	27.6%	47,751	77,700
DDS (depth 2)	30.9%	3,540	6,056,940

5.4 Comparison with State-of-the-Art Solvers

We now compare the GA using NWOX crossover and Insert mutation to the current best metaheuristic for the weighted tardiness scheduling problem with sequence-dependent setups—the VBSS-HC algorithm of Cicirello and Smith [5]. VBSS-HC is a hybrid of Value Biased Stochastic Sampling (VBSS) [5] using the Apparent Tardiness Cost with Setups (ATCS) heuristic of Lee et al [10] and Lee et al's hill climber [10]. Many of the previous best known solutions to the benchmark problem instances have been found by VBSS-HC. We also include in the results presented here a comparison with Limited Discrepancy Search (LDS) [9] and Depth-Bounded Discrepancy Search (DDS) [16] that were previously reported in [5]. Both the LDS and DDS implementations use ATCS for guidance.

Table 1 shows the average percent deviation from the best known solutions (%Δ), the number of solutions evaluated by the search on average, and the average number of search nodes generated by the search but not necessarily evaluated. For VBSS-HC, DDS, and LDS many of these generated search nodes are not solution nodes, but intermediate nodes along the search trajectory.[1] For NWOX-Insert, the average number of search nodes generated counts both the intermediate solutions after a recombination as well as the final solution after both crossover and mutation (if both operators were applied to an individual). Children that are copies of parents that did not undergo mutation or crossover are not counted among the solutions evaluated since they are not reevaluated by the algorithm. For NWOX-Insert, the table shows results for different length searches (500, 1000, 5000, and 10000 generations). For VBSS-HC, results are shown for different length searches (1000, 5000, and 10000 restarts). DDS is shown for two different depth searches.

The first observation we make from the results is that the longest search using NWOX-Insert (10000 generations) deviates the least on average from the best known solutions. While doing so, it evaluates more solutions than any other

[1]For VBSS-HC, LDS, and DDS, the average number of solutions evaluated and average number of search nodes generated are as reported by Cicirello and Smith [5]. Additionally, for VBSS-HC, LDS, and DDS, %Δ was recomputed from the raw data for the experiments of [5] using the new set of best known solutions.

Table 2: Summary of new best known solutions found by NWOX-Insert. Both the new best and the old best solutions are listed. Only problem instances for which new best solutions have been found are shown.

Instance	New	Old	Instance	New	Old
1	866	978	58	53460	55522
2	5907	6489	61	78667	79884
3	1936	2348	62	45522	47860
4	7251	8311	64	94165	96378
5	5233	5606	65	132306	134881
6	8131	8244	66	62266	64054
7	4130	4347	67	29443	34899
8	299	327	68	23614	26404
9	7421	7598	69	72238	75414
17	461	462	70	78137	81200
24	1103	1791	72	51634	56934
27	74	229	73	34859	36465
28	0	72	75	24429	30980
30	333	575	76	66656	67553
37	1757	2407	78	24030	25105
42	61855	61859	80	25212	31844
44	37584	38726	81	386782	387148
45	61573	62760	82	412721	413488
46	37112	37992	87	401049	403016
47	76629	77189	89	415433	416916
48	67633	68920	101	354906	355822
49	81220	84143	104	361718	362008
50	33877	36235	107	355537	356645
55	74342	76368	108	467651	468111
56	84614	88420			

algorithm considered except for LDS. However, it is actually doing far less work. For example, if you look at the number of search nodes generated, it generates only a fraction of the number of search nodes than does VBSS-HC—even less than the 1000 restart VBSS-HC. This is even more significant when you consider the amount of work that LDS and DDS are doing (see generated nodes column). VBSS-HC, LDS, and DDS all spend a great amount of time and effort heuristically evaluating partial solutions. While our NWOX-Insert uses problem independent genetic operators to effectively recombine above average complete solutions.

It is also worth noting that during the course of this study, NWOX-Insert was able to improve upon the best known solutions for 49 of the 120 problem instances from the benchmark set. A summary listing the new best solutions along with the previous best solutions for these 49 problem instances can be found in Table 2.

6. CONCLUSION

In this paper, we have presented a new variation of the OX operator that we call NWOX (Non-Wrapping Order Crossover). This novel crossover operator for the permutation representation of a GA strongly preserves the relative position of the alleles of the parents—to the same degree as does the original OX. However, it is designed to acknowledge that the absolute position of the alleles in the parents may also be of importance. It respects those absolute positions by not allowing wrapping of values in the left of the parents to the right end of the child permutations.

This NWOX operator is proven ideal for a problem known as weighted tardiness scheduling with sequence-dependent setups. The sequence-dependent setups of this NP-Hard scheduling problem present a challenge that only heuristic or metaheuristic approaches are currently able to handle since it is not currently possible to guarantee optimal solutions for problem instances greater than 20-30 jobs in size. The strength of NWOX in retaining order information from the parent schedules is well-suited to the sequence-dependent setup constraints; while its respect of absolute position prevents child schedules from making drastic moves of segments of jobs from the beginning to end of the schedule or vice versa. The end result is the new best metaheuristic for this computationally challenging problem. Future work will explore other appropriate applications for NWOX.

7. REFERENCES

[1] L. Adler, N. M. Fraiman, E. Kobacker, M. Pinedo, J. C. Plotnitcoff, and T. P. Wu. BPSS: a scheduling system for the packaging industry. *Operations Research*, 41:641–648, 1993.

[2] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Mahwah, NJ, USA, 1987.

[3] W. Y. Chiang, M. S. Fox, and P. S. Ow. Factory model and test data descriptions: OPIS experiments. Technical Report CMU-RI-TR-90-05, The Robotics Institute, Carnegie Mellon University, March 1990.

[4] V. A. Cicirello and S. F. Smith. Modeling GA performance for control parameter optimization. In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 235–242. Morgan Kaufmann Publishers, 8-12 July 2000.

[5] V. A. Cicirello and S. F. Smith. Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11(1):5–34, 2005.

[6] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proc. of the IJCAI*, pages 162–164, 1985.

[7] D. E. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Mahwah, NJ, USA, 1985.

[8] S.-Y. Ho and J.-H. Chen. A ga-based systematic reasoning approach for solving traveling salesman problems using an orthogonal array crossove. In *HPC '00: Proceedings of the The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 2*, pages 659–663, Washington, DC, USA, 2000. IEEE Computer Society.

[9] R. Korf. Improved limited discrepancy search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Volume 1*, pages 286–291. AAAI Press, 1996.

[10] Y. H. Lee, K. Bhaskaran, and M. Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52, 1997.

[11] A. Moraglio and R. Poli. Topological crossover for the permutation representation. In *Workshop Program of the Genetic and Evolutionary Computation Conference (GECCO2005)*, pages 332–338, Washington, D.C., USA, 25-29 June 2005. ACM Press.

[12] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley and Sons, 1993.

[13] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[14] A. K. Sen and A. Bagchi. Graph search methods for non-order-preserving evaluation functions: Applications to job sequencing problems. *Artificial Intelligence*, 86(1):43–73, September 1996.

[15] T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. A comparative study of genetic sequencing operators. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 69–76, 1991.

[16] T. Walsh. Depth-bounded discrepency search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1388–1395. Morgan Kaufmann, 1997.

[17] J.-P. Watson, C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, L. D. Whitley, and A. E. Howe. The traveling salesrep problem, edge assembly crossover, and 2-opt. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 823–834, London, UK, 1998. Springer-Verlag.