Commonality and Genetic Algorithms

Stephen Chen and Stephen F. Smith
CMU-RI-TR-96-27

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 1996

The work described in this paper was sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Material Command, USAF, under grant number F30602-95-1-0018 and the CMU Robotics Institute. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency and Rome Laboratory or the U.S. Government.

Contents

List of Figures

List of Tables

abstract

The commonality hypothesis introduced in this paper suggests that the preservation of common schemata is the central source of power in recombination operators. A commonality-based crossover operator proceeds in two steps: 1) identify the maximal common schema of two parents, and 2) complete the solution with a construction heuristic. Using this framework, two new crossover operators are proposed for sequencing problems. The first uses partial order for the basis of commonality. This operator is shown to perform well on the Traveling Salesman Problem (TSP), and it finds new best-known solutions for many Sequential Ordering Problem (SOP) instances. The second operator is based on sub-tours/edges, and it is used to demonstrate the utility of the new framework for designing hybrid genetic algorithms.

1. Introduction

A genetic algorithm (GA) has three basic features: a population of solutions, fitness-based selection, and crossover. In isolation, fitness-based selection over a population of solutions causes various building blocks (or schemata) to increase or decrease over time in direct proportion to their observed fitness. These two features provide the basis for exploiting existing knowledge during the search process. The role of crossover is to use these building blocks to explore for better solutions.

The effect of crossover has classically been analyzed from the viewpoint of a single parent. Crossover distributes the parts of each parent over two offspring. This makes it unlikely that either offspring receives "long" schemata directly from a single parent. Thus, it is suggested that the critical building blocks manipulated by crossover are "short, low-order, above-average schemata" [Gol89]. The power of crossover then comes from its ability to preserve and recombine these building blocks when assembling better solutions.

In this paper, a new view of which building blocks are the most important to the search process is proposed. Examining the effect of traditional crossover operators from the viewpoint of both parents, it can be seen that schemata common to both parents are unconditionally propagated to offspring solutions. We hypothesize that these common schemata are in fact the critical building blocks that crossover must preserve, and that the power of crossover comes from using this set of schemata as a base for exploration.

This "commonality hypothesis" leads to a new model for designing crossover operators. Having determined to keep the common components, it remains only to specify how to extend this partial solution into a complete solution. This can be done randomly, with the left-over parts of the parents (i.e. the current building blocks), or with a problem specific construction heuristic. To generalize, a commonality-based crossover operator is designed by following this framework: 1) identify the maximal common schema of two parents, and 2) complete the solution with a construction heuristic.

For traditional, positionally-dependent solution spaces, application of the framework results only in a pre-disposition to use uniform crossover. The traditional crossover operators all preserve commonality in the parents, but only uniform crossover places no restrictions on how the remaining solution space is explored. However, the implications are more significant for problems with non-standard solution spaces (e.g., sequencing problems, constrained optimization problems). Here, the design of crossover operators has often failed to exploit common solution components. For (constrained) sequencing problems, application of the commonality-based crossover framework leads to the development of two new operators: Maximum Partial Order/Arbitrary Insertion (MPO/AI) and Common Sub-Tours/Nearest Neighbor (CST/NN). Experiments will show that these are effective operators.

The remainder of this paper is divided as follows. In section 2, traditional crossover operators are reexamined using commonality. In section 3, the commonality-based crossover framework is presented. In section 4, previously developed crossover operators for sequencing problems are reviewed in light of the commonality hypothesis. In section 5, the MPO/AI operator is developed using order for the basis of commonality. An experimental analysis of the performance of MPO/AI on a range of sequencing problems is given in section 6. In section 7, the CST/NN operator, which uses common sub-tours as its basis, is developed, and experimental results for it are given in section 8. In Section 9, a discussion of broader implications is given.

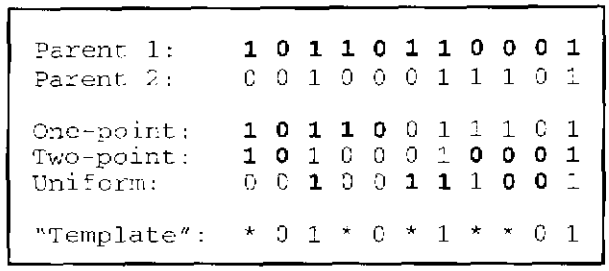

figure 1: Example of traditional crossovers. The 1's and 0's in the "template" are guaranteed to be a part of the offspring regardless of the crossover and the cut-points.

2. Traditional Crossover Operators

The power of crossover comes from striking a good balance between the exploitation and exploration of schemata. However, no attempt is made to determine which schemata are responsible for a solution's observed fitness. Thus, with the disruption of any schemata being potentially dangerous, the traditional analysis has tended to focus on the survival/disruption probabilities of various schemata. The conclusion drawn is that "short" schemata are most likely to survive, and therefore, they are the key components that drive the search process.

A different hypothesis follows if crossover is examined from the perspective of both parents. Under this view, a "template" from which the offspring are generated is revealed. (See figure 1.) This "template" is the maximal common schema of two parents. All traditional crossover operators exploit this schema without risk of disruption, and it thus provides an identifiable base from which exploration occurs. The commonality hypothesis suggests that this maximal common schema is primarily responsible for the (high) observed fitness of both parents.

Experimental results in [Sys89] show that uniform crossover out performs two-point crossover, which itself out performs one-point crossover. If the maximal common schema has n wildcard slots (*), one-point crossover explores a set with $2n$ possible offspring, two-point crossover can explore n^2-n possible offspring, and uniform crossover can explore 2^n possible offspring. Since these are supersets, the best possible offspring for two parents under uniform crossover is at least as good as that under two-point crossover which is at least as good as that under one-point crossover. This dominance relationship may explain the relative performance among the operators.

3. The Commonality-Based Crossover Framework

When designing crossover operators, the classical view of crossover's source of power (i.e., short, high performance schemata) leads to the development of operators that emphasize selection and propagation of short schemata. For the traditional representations (positionally-dependent, binary strings), these operators have also guaranteed that schemata common to both parents are not disrupted. However, for non-standard representations (e.g., spaces with positional independence and/or complex constraints), crossover operators have generally not given special attention to preserving the common schemata of both parents. It has proved more difficult to develop effective crossover operators for these non-standard problem spaces.

The commonality hypothesis identifies the schemata common to above-average parents to be the most critical for search. The function of crossover is to ensure that these schemata are propagated to new solutions, and then to build complete solutions from these building blocks. To build good solutions, effective heuristics are necessary and available in many problem domains. These observations lead to a commonality-based framework for designing crossover operators: 1) identify the maximal common schema of two parents, and 2) complete the solution with a construction heuristic.

For traditional representations, the commonality-based crossover framework suggests keeping the schemata common to the two parents, and filling in the remaining slots randomly. This is equivalent to uniform crossover. For sequence-dependent representations, following the framework leads to two new crossover operators. They are presented and evaluated later in this paper. For constrained optimization problems, a schema common to two feasible parents identifies a space with at least two feasible solutions (i.e. the parents). If a construction heuristic that can generate feasible solutions is available, it may be possible to build an effective commonality-based crossover operator around it. Finally, the framework also provides useful guidelines for incorporating high performance heuristics into hybrid genetic algorithms.

4. Sequence-based Crossover Operators

Many sequence-based crossover operators have been proposed. In the following paragraphs, a brief

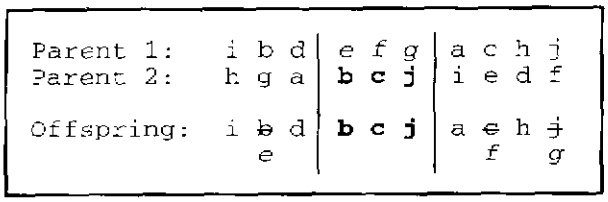

figure 2: Example of PMX. Mappings are b-e, c-f, j-g. To Parent 1, b,c,j are relocated into a sub-tour, and e,f,g are scattered.

overview of them is conducted in light of the commonality hypothesis. For the possible commonality bases of sub-tours, order, edges, and position; each operator will be examined to see which parts from the parents are kept and how the remaining solution is generated.

Partially Mapped Crossover (PMX) [GL85] is a two-point crossover. The operator takes the elements between the cut points from Parent 2. Then, it takes the elements in the first and last section from Parent 1. If an element has already been provided by Parent 2, it takes the corresponding (i.e. mapped) element from Parent 1. (See figure 2.) To Parent 1, the effect of PMX is to have one sub-tour of elements scattered in exchange for one assembled from scattered elements. To Parent 2, a sub-tour is kept, but a virtually random rearrangement occurs elsewhere.

Order Crossover (OX) [Dav85] is also a two-point crossover. Like PMX, OX takes the elements of Parent 2 between the cut points[1]. OX continues from the second cut point with the elements of Parent 1; however, if the element has been supplied by Parent 2, it is skipped (to maintain order). (See figure 3.) The spliced sub-tour of Parent 1 is no longer scattered as in PMX. However, Parent 2 again only keeps a sub-tour before a rearrangement of its remaining tour occurs.

These two operators, PMX and OX, attempt to follow the form of two-point crossover, but their effect is very dissimilar. PMX and OX do not necessarily maintain common sub-tours. Further, during the process of interleaving the chosen sub-tour into the parent to build an offspring tour, unpredictable mutations to other sub-tours occur.

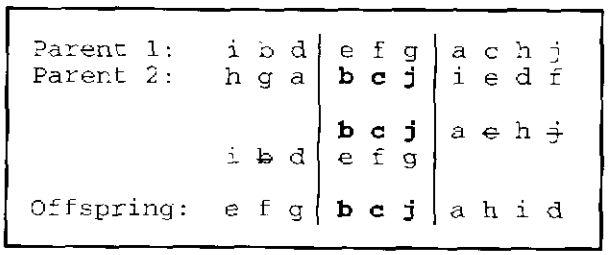

figure 3: Example of OX. To Parent 1, b,c,j are relocated into a sub-tour.

1. The role of the two parents in [Dav85] have been interchanged to highlight the similarities of OX to PMX.

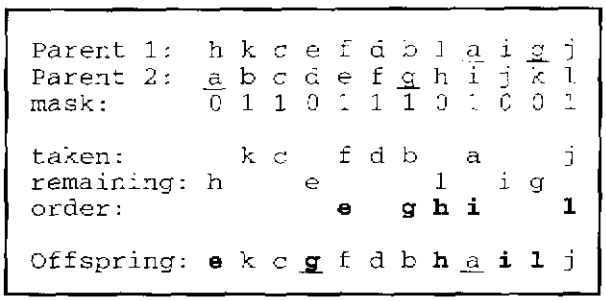

figure 4: Example of UX. The order of a and g are switched in parents and offspring.

Uniform Order-Based Crossover (UX) [Sys91] uses a uniform crossover mask. The operator takes the elements from Parent 1 in the slots where the mask has a 1. The order for the remaining elements is taken from Parent 2, and they are added in the empty slots to complete the offspring. (See figure 4.) Each parent transfers only part of its ordering information to the offspring, thus it is possible for common ordering information to be lost.

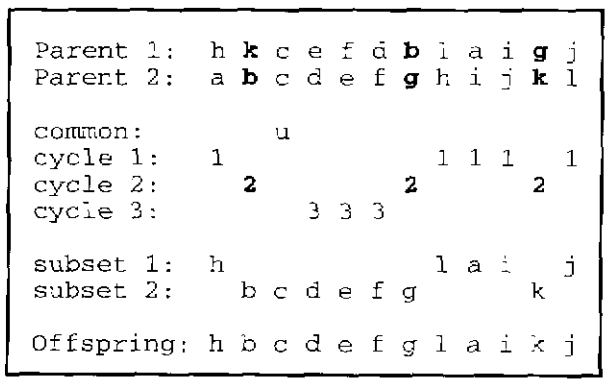

figure 5: Example of CX. Subset 1 has the elements of Parent 1 in the locations of cycle 1.

Cycle Crossover (CX) [OSH87] is like a multi-point crossover. In a cycle, the same elements are in both parents. Restricting cut-points to cycle boundaries allows normal crossover to be performed without creating an infeasible solution. (See figure 5.) The problem with CX is that it is essentially a position-based operator acting on sequence-based solutions. Cycle Crossover usually performs worse than the sub-tour based operators PMX and OX.

Edge Recombination (ER) [SMM91] is significantly different from traditional crossover operators. It uses an edge map as an intermediate

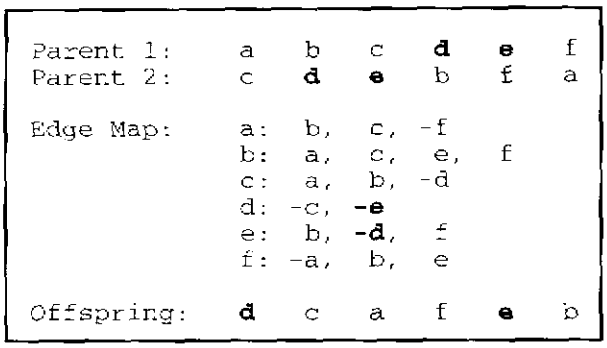

figure 6: Example of ER. Start with d (2 choices), and pick c next (randomly). Updated edge map for a: b,e,-f. Due to minus label, f is chosen. Edge e-d is lost and edge b-d is added (mutation) when f chooses e.

representation to process the parent sequences into an offspring sequence. The edge map lists the elements that are adjacent to an element (preceding or succeeding) in either of the parent tours. If an element appears twice, it is marked negative to signify a common edge. Each element is adjacent to 2, 3, or 4 other elements. To build the tour, choose the element that is negative or that has the fewest remaining adjacent elements. (See figure 6.) Under ER, an offspring receives most of the common edges and sub-tours of its two parents. However, for the n remaining elements, ER considers only 2n of the n^2 possible edges. Unlike the traditional binary representations, the uncommon parts do not span the remaining search space.

The previous operators use the sequence representation. However, a sequence can be represented by a Boolean matrix. The matrix is n x n, and element (i,j) is 1 if element j follows element i in the sequence, and it is 0 otherwise. A matrix represents a feasible solution if three constraints are satisfied.

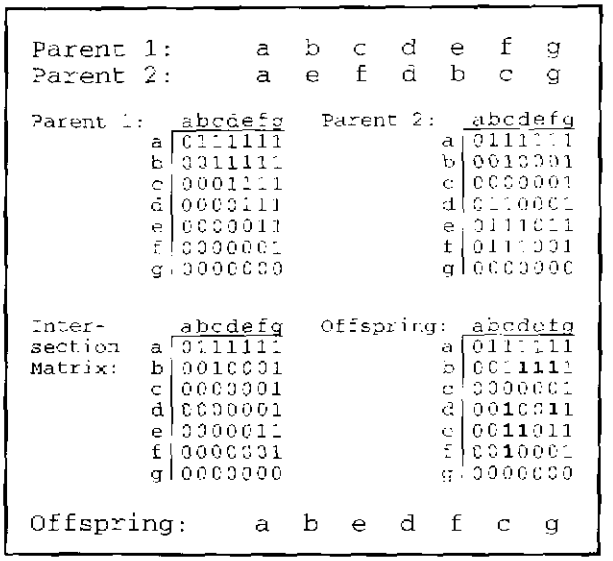

figure 7: Example of matrix intersection operator.

First, the order must be complete: there are n(n-1)/2 ones. Second, transitive order is maintained: if (i,j)=1 and (j,k)=1, then (i,k)=1. Third, there are no cycles: (i,i)=0 for all i. If two feasible matrices are intersected, a matrix satisfying the last two constraints is created. This under-constrained matrix can be completed by adding random 1's, with an analysis of row/column sums to ensure feasibility. (See figure 7.) The Matrix Intersection operator (MI) [FM91] is $O(n^2)$, and it is thus much slower than the previous operators. Further, the remaining search space after intersection is still exceptionally large, O(n!), so the random filling of 1's is incapable of effective search.

The operators discussed to this point are all generic and "blind"--they use no auxiliary information, only the total tour length. This allows the operators to be used directly on other problems, including

TSP Instance	Size	start	PMX	OX	UX	CX	ER	MI
d198	198	+ 978 %	+ 353 %	+ 110 %	+ 231 %	+ 508 %	+ 267 %	+ 918 %
lin318	318	+ 1191 %	+ 632 %	+ 383 %	+ 1074 %	+ 843 %	+ 560 %	+ 1171 %
fl417	417	+ 3724 %	+ 1886 %	+ 992 %	+ 3311 %	+ 2447 %	+ 1607 %	+ 3646 %
pcb442	442	+ 1321 %	+ 760 %	+ 526 %	+ 1221 %	+ 972 %	+ 723 %	+ 1300 %
u574	574	+ 1626 %	+ 945 %	+ 714 %	+ 1517 %	+ 1196 %	+ 903 %	+ 1608 %
average		+ 1768 %	+ 915 %	+ 545 %	+ 1471 %	+ 1193 %	+ 812 %	+ 1729 %

Table 1: Performance of sequence-based crossover operators on five TSP instances taken from TSPLIB. Results are for a steady-state GA with a population size of 1000 run for 250 generations. (MI run with a population of 200 for 40 generations.) Values are percent distance from known optimum for average of 5 runs.

those that might not have auxiliary information (e.g. scheduling). Unfortunately, the results are quite poor. (See Table 1.) The following multi-operator approaches incorporate heuristics that use auxiliary information (i.e. edge weights).

A Greedy Crossover (GX) [GGR85] operator that uses edge weights to determining which parts should be taken from each parent has been proposed. The most effective version of GX starts with a random element, and builds the tour by picking the shortest edge from either parent that does not introduce a cycle. If all edges from the parents cause cycles, the edge to the nearest available element is selected instead. If a common edge is not the shortest edge connected to an element, it can be lost.

A hybrid genetic algorithm applies a local optimizer to each solution after crossover. The solutions created by GX often have crossing edges, so 2-Opt can be added to correct these mistakes. The GX-2-Opt hybrid GA was developed by [JSG89]. Gen2-Opt is another 2-Opt based hybrid GA that has been developed by [UPL90]. However, neither GA is built on a crossover operator that focuses on maintaining common schemata from the parents.

The A-team approach of [TS92] incorporates a large variety of optimization methods into an asynchronous organization. In the context of genetic crossover operators, the most interesting part of the A-team approach is a new deconstruction/reconstruction (Dec/Rec) operator. This operator has the form of a commonality-based crossover operator. The deconstruction step creates a sub-tour by connecting the common edges of two parents in the order and orientation of Parent 1. (See figure 8.) The reconstruction step uses the Arbitrary Insertion[1] (AI) construction heuristic to complete the sub-tour. However, the deconstruction procedure is edge based, and the reconstruction procedure is order based. The result of this mismatch is that the common edges are not necessarily preserved.

```
Parent 1:  a b c d e f g h i j k l
Parent 2:  a c e d k i j f h g l b

edges:     a-b   d-e   g-h   i-j

Sub-tour:  a b d e g h i j
```

figure 8: Example of deconstruction. Edges a-b, d-e, and g-h have reverse orientation.

1. Starting with a sub-tour, pick an arbitrary element not in the sub-tour, and insert it in the least cost position. Repeat until all elements have been inserted.

5. Maximum Partial Order

To build an operator under the commonality-based crossover framework, it is necessary to select a basis of commonality and a matching construction heuristic. For sequencing problems, the choices for the basis of commonality are sub-tours, order, edges, and positions. The first new crossover operator uses order as its basis of commonality. Insertion heuristics preserve order, so any partial order that can be extended by insertion into both parents may be considered to be common to the two parents. The longest such partial order is the Maximum Partial Order (MPO).

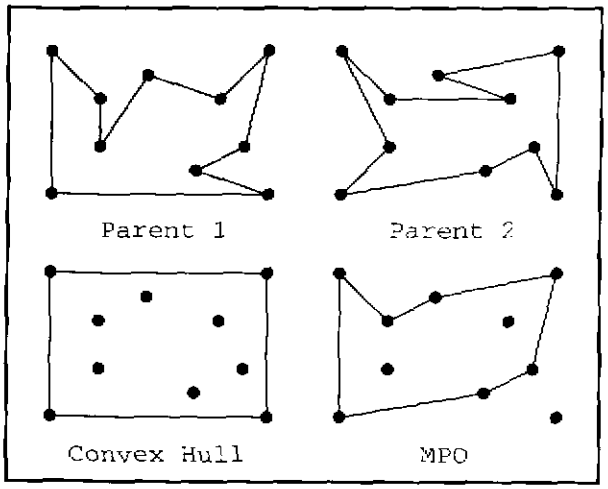

figure 9: Example of "shape" from convex hull and MPO.

The MPO constitutes a notion of "shape". While good TSP tours appear "smooth", poor TSP tours appear "jagged". A partial order eliminates much of this "noise" and creates a good tour outline. (See figure 9.) AI is a good match with MPO since the performance of AI is improved when it is given a good tour outline. For example, AI started from the convex hull (CH) develops tours that are about 1% better than those developed when AI is started from three random elements [Rei94]. Ideally, the MPO of two parent tours eventually becomes an outline that AI can generate the optimum tour from.

- Represent parents by Boolean matrices
- Intersect matrices
- Sum columns to get each element's predecessors
- Build partial order graph
  - Find element with fewest predecessors
  - Attach element to the predecessor with the most ordered predecessors
- Find longest path in graph

figure 10: Pseudo-code for MPO.

The pseudo-code for generating the Maximum Partial Order of two parents is given in figure 10. It is assumed that all tours have the same first element and the same orientation[1]. The intersection matrix has element (i,j)=1 if and only if element j follows element i in both of the parents. The column sums give the number of predecessors for each element that are common to both parents.

The partial order graph starts with the segment start element, and at each node, it gives the longest partial order for that element. (All ties are broken randomly.) When all elements have been added, the longest path in the graph is the Maximum Partial Order. (See figure 11.)

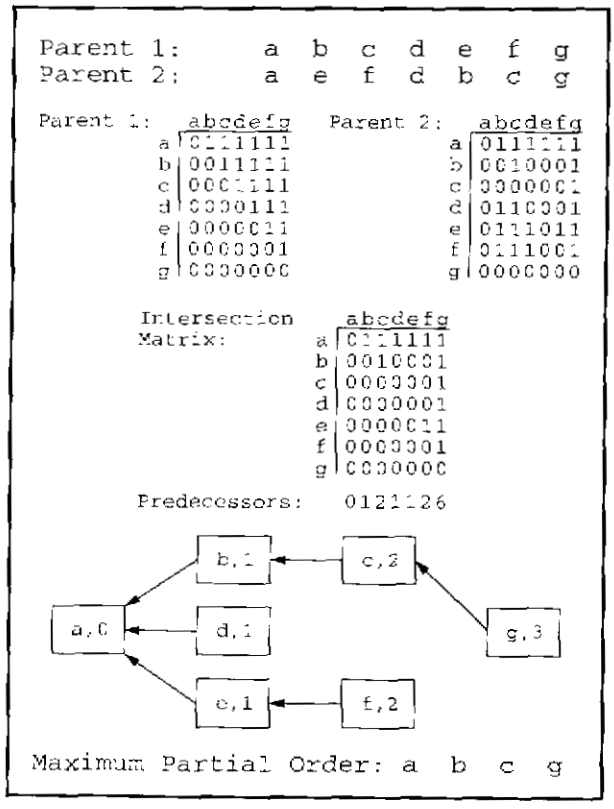

figure 11: Example of MPO. Element g is after all elements, but elements c and f have the most ordered predecessors (2).

1. Three evenly spaced convex hull elements (e.g. a,b,c) define the orientation of all tours (i.e., a...b...c...). These elements are our segment start elements.

6. Experimental Results for MPO/AI

6.1 Traveling Salesman Problem

The Maximum Partial Order/Arbitrary Insertion operator was designed for the Traveling Salesman Problem (TSP). The objective in the TSP is to find the shortest Hamiltonian cycle in a complete graph of n elements with symmetric edge weights, i.e. $d(c_i, c_j) = d(c_j, c_i)$. Formally, given n elements c_i with edge weights $d(c_i, c_j)$, find a sequence π of the n elements such that the sum is minimized.

$$\min_{\pi} \sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

The MPO/AI operator has been implemented in a steady-state genetic algorithm (one-at-a-time reproduction, worst member removed, no duplicate solutions) [WS90]. The population size was set to 400 solutions and the stop condition of 10 generations without improving the best overall solution was used. The population was seeded with 400 convex hull/AI solutions. Since AI generates solutions of low diversity, it was necessary to slow convergence by selecting only one parent by 2-tournament[2] (biased to good solutions) and the other randomly. Results are given as percent distance above the known optimum for the average of 5 runs. All experiments were run on a SPARC 20, and distances were stored in full matrices.

The MPO/AI operator was tested on 5 TSP instances[3] taken from the TSPLIB library[4]. On average, the final MPO/AI tour is 4.5% better than the best CH/AI tour. (See Table 2.) This reflects MPO's ability to extract the "shape" of good tours. Compared against the Lin-Kernighan heuristic procedure [LK73] (widely regarded as the best TSP heuristic), the GA's solution quality is more consistent and better on average, but in the time required to run the GA once, many runs of LK could be performed and the best run would likely be better.

2. Select two solutions randomly from the population. Then, selecting one based on a criterion from these two creates a linear rank-based probability distribution for selection on that criterion.
3. 5 smallest TSP instances with LK results in [Rei94].
4. Available by anonymous ftp: ftp://elib.zib-berlin.de/pub/mp-testdata/tsp

TSP Instance	Size	Avg. Best CH/AI	Avg. Best MPO/AI	Runtime (min)	Avg. LK (12 runs)	Best LK (12 runs)
d198	198	+ 3.05 %	+ 0.95 %	3	+ 1.54 %	+ 0.60 %
lin318	318	+ 6.04 %	+ 0.63 %	13	+ 1.79 %	+ 0.69 %
fl417	417	+ 1.91 %	+ 0.57 %	15	+ 2.19 %	+ 0.51 %
pcb442	442	+ 8.97 %	+ 1.84 %	37	+ 1.80 %	+ 1.11 %
u574	574	+ 8.45 %	+ 2.20 %	74	+ 1.96 %	+ 0.93 %
average		+ 5.68 %	+ 1.20 %		+ 1.86 %	+ 0.77 %

Table 2: Comparison of MPO/AI and LK [Rei94] on 5 TSP instances.

The weakness of MPO/AI is that AI does not generate enough diversity of relative ordering schemata. Thus, the GA converges before the optimum solution is found. This convergence occurs in a relatively small number of generations (~25), but the $O(n^2)$ complexity of the operator still causes high runtimes.

Another experiment was conducted to measure the importance of properly matching the construction heuristic to the basis of commonality. Holding the construction heuristic constant, AI was also "matched" to Matrix Intersection [FM91] and Deconstruction [TS92]. The ordering information in MI that is not part of the MPO is coincidental and it handicaps the performance of AI. (See figure 12.) Consequently, the results of MI/AI are slightly worse for the TSP. (See Table 3.) AI does not necessarily preserve the edges of Deconstruction, so information about good edges is not accumulated. Further, deconstruction can flip edge orientation (see figure 8), so the tour outline it provides

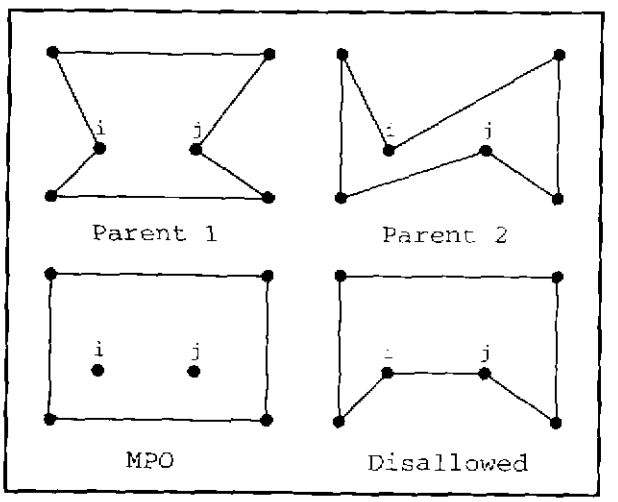

figure 12: Element i precedes j in both parents, but AI might determine that j should precede i.

AI is of little benefit. These results demonstrate the importance of properly matching the two parts of a crossover operator. The construction heuristic should be aided by, but not restricted by nor destructive to, the common elements.

TSP Instance	Avg. Best CH/AI	Avg. Best MPO/AI	Runtime (min)	Avg. Best MI/AI	Runtime (min)	Avg. Best Dec/Rec	Runtime (min)
d198	+ 3.05 %	+ 0.95 %	3	+ 1.14 %	4	+ 1.31 %	1
lin318	+ 6.04 %	+ 0.63 %	13	+ 1.42 %	15	+ 3.38 %	2
fl417	+ 1.91 %	+ 0.57 %	15	+ 0.61 %	18	+ 0.52 %	7
pcb442	+ 8.97 %	+ 1.84 %	37	+ 3.18 %	47	+ 4.20 %	9
u574	+ 8.45 %	+ 2.20 %	74	+ 2.63 %	73	+ 4.01 %	13
average	+ 5.68 %	+ 1.24 %		+ 1.80 %		+ 2.68 %	

Table 3: Comparison of MPO/AI, MI/AI, and Deconstruction/Reconstruction on 5 TSP instances.

6.2 Sequential Ordering Problem

The MPO reduces the search space of AI. A similar reduction of the search space occurs if ordering constraints are given, as in the Sequential Ordering Problem (SOP). Specifically, $d(c_i, c_j) = -1$ if element j must precede (not necessarily immediately) element i. The objective in the SOP is to find the shortest Hamiltonian path subject to satisfying all ordering constraints. The SOP is based upon an Asymmetric Traveling Salesman Problem (ATSP), thus edge weights may be asymmetric. Formally, given n elements c_i with edge weights $d(c_i, c_j)$, find a sequence π of the n elements such that the sum is minimized and all ordering constraints are satisfied.

$$\min_{\pi} \sum_{k=1}^{n-1} d(c_{\pi(k)}, c_{\pi(k+1)})$$

s.t. if $d(c_i, c_j) = -1$
then $l < k$
for $\pi(l) = j$, $\pi(k) = i$

The SOP is a useful model for production planning [Esc88], single-vehicle routing problems with pick-up and delivery constraints [PT91][Sav90], and transportation problems within flexible manufacturing systems [Asc96].

The MPO/AI operator was run on 16 SOP instances[1]. Since Arbitrary Insertion uses "shape" which can be distorted by asymmetry, AI is less effective and less consistent. Thus, the GA parameters were increased to 500 solutions/20 generations, and both parents were selected by 2-tournament. The population was seeded with 500 AI solutions. The higher population size and stop condition are largely overkill for the smaller SOP instances, but they lead to significantly better final solutions for larger SOPs (the "rbg" instances). (See Table 4.) The overall runtime may seem high, but solutions that are below the previous best-known bounds are found quickly. In general, the genetic

1. To handle constraints, AI was trivially modified to construct only feasible solutions.

SOP Instance	Size	Constraints	Avg. Best AI	Avg. Best MPO/AI	Overall Best MPO/AI	Previous Bounds	Time to Bound (s)	Runtime (min)
ft70.1	71	17	41809	39615	39545	**39313**	NA	2
ft70.2	71	35	43485	40435	**40422**	[39739, 41778]	11	2
ft70.3	71	68	46731	42558	**42535**	[41305, 44732]	6	2
ft70.4	71	86	55982	53583	**53562**	[52269, 53882]	5	2
kro124p.1	101	25	45758	40996	**40186**	[37722, 42845]	11	4
kro124p.2	101	49	49056	42576	**41677**	[38534, 45848]	9	4
kro124p.3	101	97	63768	51085	**50876**	[40967, 55649]	7	4
kro124p.4	101	131	87975	76103	**76103**	[64858, 80753]	4	4
rbg323a	325	2412	3466	3161	**3157**	[3136, 3221]	361	46
rbg341a	343	2542	3184	2603	**2597**	[2543, 2854]	131	64
rbg358a	360	3239	3165	2636	**2599**	[2518, 2758]	934	102
rbg378a	380	3069	3420	2843	**2833**	[2761, 3142]	120	147
ry48p.1	49	11	16602	15813	**15805**	[15220, 15935]	4	1
ry48p.2	49	23	18071	16676	**16666**	[15524, 17071]	3	1
ry48p.3	49	42	22074	19905	**19894**	[18156, 20051]	15	1
ry48p.4	49	58	32591	31446	31446	[29967, 31446]	9	1

Table 4: Performance of MPO/AI on 16 SOP instances. Overall best final tours in bold improve previous best known bound.

ATSP Instance	Size	Population size = 400			Population size = 10		
		Avg. Best AI	Avg. Best MPO/AI	Runtime (min)	Avg. Best AI	Avg. Best MPO/AI	Runtime (min)
rbg323	323	+ 18.37 %	+ 16.30 %	24	+ 20.89 %	+ 9.22 %	3
rbg358	358	+ 26.50 %	+ 22.05 %	28	+ 33.31 %	+ 7.79 %	4
rbg403	403	+ 5.83 %	+ 4.10 %	25	+ 8.13 %	+ 1.61 %	3
rbg443	443	+ 6.64 %	+ 4.98 %	35	+ 8.22 %	+ 1.68 %	4
average		+ 14.34 %	+ 11.86 %		+ 17.64 %	+ 5.08 %	

Table 5: Performance of MPO/AI for two different population sizes on 4 ATSP instances.

algorithm finds better solutions than a branch and cut algorithm[1] [Asc96] in roughly 2% of the CPU time.

6.3 Asymmetric Traveling Salesman Problem

For completeness, the MPO/AI operator was also run on 4 ATSP instances[2]. Two GAs were run with different parameter settings: 400 solutions/10 generations and 10 solutions/50 generations. Each population was seeded by AI. In an unusual result, better results are obtained for the smaller population size. (See Table 5.) This result might be caused by the inconsistency in AI for asymmetric problems. A large population likely contains many diverse solutions which have little in common. However, a small population may quickly converge to a reduced search space where some local optimization is possible.

6.4 Summary

The Maximum Partial Order of two parent solutions represents a common "shape" or "structure". The Arbitrary Insertion construction heuristic uses this shape as an outline to develop good tours for the TSP. Comparatively, the MPO/AI operator performs very well on the SOP and quite poorly on the ATSP. In the ATSP, tour orientation cannot be fixed. Using only one segment, MPO returns noise if the parents are of opposite orientation. In the SOP, orientation is again fixed, and MPO performs much better. More importantly, MPO is based on order, so the added ordering constraints provide structure in the SOP. In contrast, methods based on edge representations (like many integer programming formulations) have their search space convoluted by ordering constraints. On the TSP and ATSP, branch and cut algorithms can often find the optimum solutions.

1. In this study, the program was given 300 minutes on a SPARC 2; 1200 minutes for the "rbg" instances.
2. In this case, only one segment is used for MPO since no tour orientation can be fixed a priori.

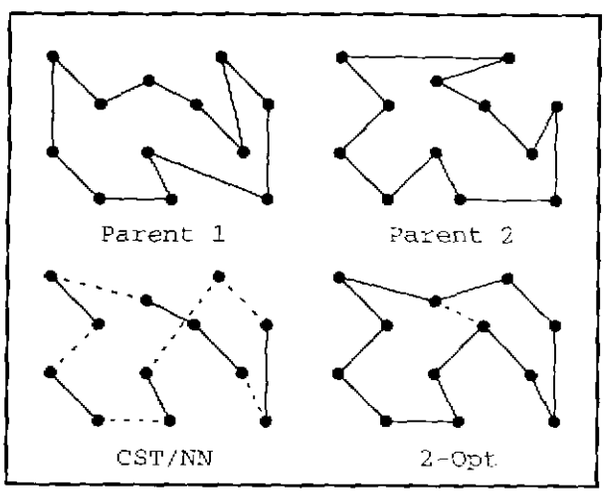

figure 13: Example of CST/NN and CST/NN-2-Opt. One common edge is lost during 2-Opt.

7. Common Sub-Tours

Order is just one possible basis of commonality for sequencing problems. A second operator was designed to take all the Common Sub-Tours (CST), and thus all the common edges, from two parents. Starting with a random common sub-tour, the remaining elements and sub-tours are connected with the Nearest Neighbor (NN) heuristic. (See figure 13.)

8. Experimental Results for CST/NN

8.1 Standard Genetic Algorithms

The CST/NN operator was run on the same 5 TSP instances introduced earlier. The GA was seeded with solutions obtained by starting NN from each

TSP Instance	Avg. Best NN Tour	Avg. Best CST/NN Tour	Common Edges	Avg. Best GX Tour	Common Edges	Avg. Best ER Tour	Common Edges
d198	+ 12.42 %	+ 5.13 %	100.0 %	+ 10.72 %	99.8 %	+ 10.53 %	97.4 %
lin318	+ 17.06 %	+ 9.16 %	100.0 %	+ 16.95 %	100.0 %	+ 17.06 %	97.6 %
fl417	+ 16.92 %	+ 13.22 %	100.0 %	+ 10.66 %	100.0 %	+ 12.54 %	98.6 %
pcb442	+15.17 %	+ 9.64 %	100.0 %	+ 12.11 %	99.0 %	+ 15.17 %	97.3 %
u574	+ 19.92 %	+ 11.36 %	100.0 %	+ 15.98 %	99.9 %	+ 19.38 %	98.1 %
average	+ 16.30 %	+ 9.70 %	100.0 %	+ 13.28 %	99.7 %	+ 14.94 %	97.8 %

Table 6: Comparison of CST/NN, GX, and ER on 5 TSP instances.

element, and the stop condition was 20 generations. The population is seeded with Nearest Neighbor to isolate the effect of CST in the crossover operator. If random solutions are used to seed the population, two parents from this initial population can be expected to have only one edge in common. Thus, the first generation of new solutions will be similar to NN solutions anyway.

The Nearest Neighbor heuristic selects many good edges, but it myopically "paints itself into a corner" and then must select a poor edge. The final CST/NN tours average a disappointing 10% from optimum. (See Table 6.) Note, however, that although Greedy Crossover [GGR85] implicitly uses NN and maintains most of the common edges, it performs over 3% worse. Two NN tours are likely to have the same good edges, but different poor edges. To minimize disruption, the GX operator forces itself to take these poor edges, thus handicapping NN. If the above-average schemata are those that are common to above-average solutions, the converse implies that the uncommon schemata may be below average.

The Edge Recombination [SMM91] operator performs worse than both CST/NN and GX. The effectiveness of ER is limited by two drawbacks. First, the operator incorporates no heuristics; and second, the operator also forces itself to take poor edges if they exist in either parent. An interesting observation is that ER maintains fewer common edges than GX--even though ER was developed to maintain common edges, and GX wasn't. The problem is that ER's first focus is to use edges from either parent. At the start, or whenever no edges are available from the current element, ER takes the element with the fewest remaining edges. If this element is internal to a common sub-tour, selecting it will cause one common edge to be lost.

8.2 Hybrid Genetic Algorithms

The mistakes of NN can be improved by 2-Opt. (See figure 13.) Like the GX-2-Opt hybrid GA of [JSG89], 2-Opt was added to CST/NN to make CST/NN-2-Opt. This hybrid GA was run on the same 5 TSP instances. The GA parameters were 400 solutions/10 generations. The population was

TSP Instance	Avg. Best 2-Opt	Avg. Best CST/NN-2-Opt	Kept By 2-Opt	Avg. Best GX-2-Opt	Kept By 2-Opt	Avg. Best ER-2-Opt	Kept By 2-Opt
d198	+ 3.06 %	+ 0.87 %	89.9 %	+ 1.12 %	87.1 %	+ 1.31 %	85.9 %
lin318	+ 5.16 %	+ 0.31 %	93.5 %	+ 0.77 %	92.1 %	+ 2.34 %	90.8 %
fl417	+ 2.54 %	+ 1.16 %	86.6 %	+ 1.19 %	83.2 %	+1.53 %	80.7 %
pcb442	+ 7.29 %	+ 1.22 %	91.1 %	+ 2.28 %	87.9 %	+ 4.07 %	83.8 %
u574	+ 8.26 %	+ 2.68 %	89.7 %	+ 3.66 %	87.5 %	+ 5.01 %	85.9 %
average	+ 5.26 %	+ 1.25 %	90.2 %	+ 1.80 %	87.6 %	+ 2.85 %	85.4 %

Table 7: Comparison of CST/NN-2-Opt, GX-2-Opt, and ER-2-Opt on 5 TSP instances.

TSP Instance	Size	Avg. Best CST/NN-2-Opt	Avg. Best Gen2-Opt	Total 2-Opt Calls
fin318	318	+ 1.35 %	+ 2.02 %	390
pcb442	442	+ 1.65 %	+ 3.02 %	720
att532	532	+ 1.61 %	+ 2.99 %	954
average		+ 1.54 %	+ 2.68 %	

Table 8: Comparison of CST/NN-2-Opt and Gen2-Opt [UPL90] on 3 TSP instances.

seeded with 400 random solutions each optimized by 2-Opt. For these tests, the CST/NN-2-Opt hybrid GA finds tours that are about 0.5% better than GX-2-Opt. (See Table 7.)

Returning the focus to commonality, some common edges can be lost during the 2-Opt process. However, over 90% of the CST edges are kept after 2-Opt during the first generation of CST/NN-2-Opt. The number of lost (disrupted) common edges is a measure of how far the crossover solution is from a 2-Opt minima. Common edges are disrupted 2.6% more often in GX-2-Opt, so GX is less effective than CST/NN in guiding 2-Opt. The results of ER-2-Opt also follow this trend--more disruption, lower solution quality.

CST/NN-2-Opt has also been compared with Gen2-Opt [UPL90]. Allowing the CST/NN-2-Opt GA to use the same number of 2-Opt calls used by the Gen2-Opt GA, a population size of 28 worked best. On the 3 common TSP instances we had available (in matrix form), CST/NN-2-Opt performs significantly better. (See Table 8.)

9. Discussion

The current analysis of crossover suggests that "short, low-order, above-average schemata" are the critical building blocks manipulated during the search process. The resulting view is that "... a genetic algorithm seek[s] near optimal performance through the juxtaposition of [these] building blocks" [Gol89], and "Crossover operators, viewed in the abstract, are operators that combine subparts of the two parent chromosomes to produce new children" [Dav91].

The commonality hypothesis suggests that the schemata common to above-average solutions are the critical building blocks. The traditional positionally-dependent, binary string crossover operators support both interpretations of where crossover gains its power. However, it may be possible that the preservation of common schemata provides a base that is necessary for the juxtaposition/combination of short, low-order schemata to be productive.

The commonality hypothesis leads to a new general framework for designing crossover operators. This framework covers many different problem domains, solution representations, and constraints. Further, auxiliary evaluation information is easily incorporated through the use of construction heuristics. These are often difficult issues under the current crossover model.

More generally, reserving solution parts from trial to trial is a natural aspect of any iterative optimization procedure. Each successive solution has a part kept from the previous trial and a part added by an operator. A typical local-search operator identifies a small part of the solution which can be changed to yield a better solution. Instead of identifying what should be changed in one solution, crossover uses two solutions to identify what is worth keeping.

Hybrid genetic algorithms often locally optimize each solution after crossover. Crossover accelerates the search because it "... pays off when intermediate solutions are of higher quality" [UPL90]. However, a genetic algorithm with a crossover operator that randomly combines subparts of two parents is not significantly different from random multi-start. Alternatively, a crossover operator designed to maintain commonality for the basis of the local optimizer can improve the performance of a hybrid genetic algorithm.

References

[Asc96] N. Ascheuer. (1996) *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. ZIB Technical Report TR 96-3.

[Dav85] L. Davis. (1985) "Applying Adaptive Algorithms to Epistatic Domains." In *Proc. International Joint Conference on Artificial Intelligence.*

[Dav91] L. Davis. (1991) *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

[Esc88] L.F. Escudero. (1988) "An Inexact Algorithm for the Sequential Ordering Problem." In *European Journal of Operations Research*, 37:236-253, 1988.

[FM91] B. Fox and M.B. McMahon. (1991) "An Analysis of Reordering Operators for Genetic Algorithms." In *Foundations of Genetic Algorithms.* G. Rawlins, ed. Morgan Kaufmann.

[GGR85] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. (1985) "Genetic Algorithms for the Traveling Salesman Problem." In *Proc. of an International Conference on Genetic Algorithms.*

[GL85] D. Goldberg and R. Lingle. (1985) "Alleles, loci, and the Traveling Salesman Problem." In *Proc. International Conference on Genetic Algorithms and their Applications.*

[Gol89] D. Goldberg. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

[JSG89] P. Jog, J.Y. Suh, and D. Van Gucht. (1989) "The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem." In *Proc. Third International Conference on Genetic Algorithms.*

[LK73] S. Lin and B.W. Kernighan. (1973) "An Efficient Heuristic Algorithm for the Traveling Salesman Problem." In *Operations Research*, 21:498-516, 1973.

[OSH87] I. Oliver, D. Smith, and J. Holland. (1987) "A Study of Permutation Crossover Operators on the Traveling Salesman Problem." In *Proc. Second International Conference on Genetic Algorithms and their Applications.*

[PT91] W. Pulleyblank and M. Timlin. (1991) *Precedence Constrained Routing and Helicopter Scheduling: Heuristic Design.* IBM Technical Report RC17154 (#76032).

[Rei94] G. Reinelt. (1994) *The Traveling Salesman: Computational Solutions for TSP Applications.* Springer-Verlag

[Sav90] M.W.P. Savelsbergh. (1990) "An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems." In *European Journal of Operations Research*, 47:75-85, 1990.

[SMM91] T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. (1991) "A Comparison of Genetic Sequencing Operators." In *Proc. Fourth International Conference on Genetic Algorithms.*

[Sys89] G. Syswerda. (1989) "Uniform Crossover in Genetic Algorithms." In *Proc. Third International Conference on Genetic Algorithms.*

[Sys91] G. Syswerda. (1991) "Order-Based Genetic Algorithms and the Graph Coloring Problem." In *Handbook of Genetic Algorithms*, L. Davis, ed. Van Nostrand Reinhold.

[TS92] S. Talukdar and P. de Souza. (1992) "Scale Efficient Organizations." In *Proc. 1992 IEEE International Conference on Systems, Man and Cybernetics.*

[UPL90] N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, H.-J. Bandelt, E.H.L. Aarts. (1990) "Improving TSP Exchange Heuristics by Population Genetics." In *Parallel Problem Solving from Nature.*

[WS90] D. Whitley and T. Starkweather. (1990) "GENITOR II: A distributed Genetic Algorithm." In *Journal of Experimental and Theoretical Artificial Intelligence*, 2:189-214, 1990.