Designing a Unique Single Point Cross Over Method

Thesis Project
Student Richard Phillip Wilson
Advisor : Dr. Clinton Lee
Professor: Abdollah Homaifar

Introduction

The idea behind GA´s is to extract optimization strategies nature uses successfully - known as Darwinian Evolution - and transform them for application in mathematical optimization theory to find the global optimum in a defined phase space.

One could imagine a population of individual "explorers" sent into the optimization phase-space. Each explorer is defined by its genes, what means, its position inside the phase-space is coded in his genes. Every explorer has the duty to find a value of the quality of his position in the phase space. (Consider the phase-space being a number of variables in some technological process, the value of quality of any position in the phase space - in other words: any set of the variables - can be expressed by the yield of the desired chemical product.) Then the struggle of "life" begins. The three fundamental principles are

1.Selection
2.Mating/Crossover
3.Mutation

Only explorers (= genes) sitting on the best places will reproduce and create a new population. This is performed in the second step (Mating/Crossover). The "hope" behind this part of the algorithm is, that "good" sections of two parents will be recombined to yet better fitting children. In fact, many of the created children will not be successful (as in biological evolution), but a few children will indeed fulfill this hope. These "good" sections are named in some publications as building blocks.

Now there appears a problem. Repeating these steps, no new area would be explored. The two former steps would only exploit the already known regions in the phase space, which could lead to premature convergence of the algorithm with the consequence of missing the global optimum by exploiting some local optimum.

The third step - the Mutation ensures the necessary accidental effects. One can imagine the new population being mixed up a little bit to bring some new information into this set of genes.

Whereas in biology a gene is described as a macro-molecule with four different bases to code the genetic information, a gene in genetic algorithms is usually defined as a bitstring (a sequence of b 1´s and 0´s). (1)

2.Genetic Algorithms

Initial Population

As described previously, a gene is a string of bits. The initial population of genes (bitstrings) is usually created randomly. The length of the bitstring is depending on the problem to be solved.

Selection

Selection means to extract a subset of genes from an existing (in the first step, from the initial -) population, according to any definition of quality. In fact, every gene must have a meaning, so one can derive any kind of a quality measurement from it - a "value". Following this quality "value" (fitness), Selection can be performed
e.g. by Selection proportional to fitness:

1.Consider the population being rated, that means: each gene has a related fitness. The higher the value of the fitness, the better.
2.The mean-fitness of the population will be calculated.
3.Every individuum (=gene) will be copied as often to the new population, the better it fitness is, compared to the average fitness. E.g.: the average fitness is 5.76, the fitness of one individuum is 20.21.
4.Following this steps, one can prove, that in many cases the new population will be a little smaller, than the old one. So the new population will be filled up with randomly chosen individual from the old population to the size of the old one.

Remember, that there are a lot of different implementations of these algorithms. For example the Selection module is not always creating constant population sizes. In some implementations the size of the population in dynamic. Furthermore, there exist a lot of other types of selection algorithms (the most important ones are: Proportional Fitness, Binary Tournament, Rank Based)..

Mating/Crossover

```
                 Parents

1    1    0    0 | 1    0    1    1    1
0    0    1    0 | 0    1    1    0    1
                ^
                 Children
1    1    0    0   0    1    1    0    1
0    0    1    0   1    0    1    1    1
```

Fig.2. Crossover

The next steps in creating a new population are the Mating and Crossover: As described in the previous section there exist also a lot of different types of Mating/Crossover. One easy to understand type is the random mating with a defined probability and the b_nX crossover type. This type is described most often, as the parallel to the Crossing Over in genetics is evident:

1. PM percent of the individua of the new population will be selected randomly and mated in pairs.
2. A crossover point (see fig.2) will be chosen for each pair
3. The information after the crossover-point will be exchanged between the two individua of each pair.

In fact, more often a slightly different algorithm called b_uX is used. This crossover type usually offers higher performance in the search.

1. PM percent of the individua of the new population will be selected randomly and mated in pairs.
2. With the probability PC, two bits in the same position will be exchanged between the two individua. Thus not only one crossover point is chosen, but each bit has a certain probability to get exchanged with its counterpart in the other gene. (This is called the uniform operator)

Mutation

0	1	0	1	0	1	1
M					M	
1	0	0	1	0	0	0

Fig.3. Mutation

The last step is the Mutation, with the sense of adding some effect of exploration of the phase-space to the algorithm. The implementation of Mutation is - compared to the other modules - fairly trivial: Each bit in every gene has a defined Probability P to get inverted. (1)

3.Motivation of my method

The search space for the traveling salesman problem is the set of permutations of the cities. The most natural way to represent a tour is through path representation, where the cities are listed in the order in which they are visited . As an example of path representation, assume that there are six cities: {1, 2, 3, 4,5,6}. The tour denoted by {1 2 3 4 5 6) would be interpreted to mean that the salesman visits city 1first, city 2 second, city 3 third ,..., returning to city 1 from city 6. Although this representation seems natural enough, there are at least two drawbacks to it. The first is that it is not unique. For example, (2 3 4 5 6 1) and (3 4 5 6 1 2) actually represents the same tour as (1 2 3 4 5 6); that is, the representation is unique only as far as the direction of traversal – clockwise or counterclockwise- and the originating city. This representational ambiguity generally confounds the GA. The second drawback is that a simple crossover operator could fail to produce legal tours. For example, The following strings with cross site 3 fail to produce legal tours. For example, the following strings with the cross site 3 fail to produce legal tours.(2)

Before crossover	(1	2	|	3		4	5	6)
	(2	4	|	5		6	3	1)
					^			
After crossover	(1	2		3		6	3	1)
	(2	4		5		4	5	6)

The two routes created after crossover are illegal because both go to at least one city more than once. (2)

4.Currently used cross over operators which eliminate repetition

Goldbergs's partially-mapped crossover (PMX)

The PMX uses a series of swapping to avoid duplication of cities. For example, there are two tours (3):

Tour 1	a	b	c		d	|	e	f		g	|	h
Tour 2	b	d	f		a	|	g	h		c	|	e
				^					^			
One of the offsprings:	b	a	h		d		e	f		c		g

Davis' order crossver(OX)

OX tries to maintain the original city order in the parents in the condition of no duplication. For the same example as above(3)

```
      Tour 1        a     b     c  |  d     e     f  |  g     h
      Tour 2        b     d     f  |  a     g     h  |  c     e
                                   ^                 ^
One of the offsprings: a  g     h     d     e     f     c     b
```

Oliver's cycle crossover (CX)

CX creates a legal offspring where every position is occupied by a corresponding element from one of the parents by finding and combing the cycles in the parents. For the same example as above (3)

```
      Tour 1        a     b     c     d     e     f     g     h
      Tour 2        b     d     f     a     g     h     c     e
                                   ^                 ^
One of the offsprings:
                    a     b     f     d     g     h     c     e
```

5. Proposed crossover method

In my proposed method of crossover I use grouping in order to limit the search of the repeated and missing cities of each child after an initial single point crossover of the initial parent routes. I then replace the repeated cities randomly with the missing cities found as a product of selection and crossover. Mutation has yet to be included. The program written is designed for routes consisting of 2^n cities. Test included 16 cities.

When using 16 cities the first step is to create a population of routes with each route consisting of cities 0-15 (decimal numbers are used to represent cities). For my example I used a population of 512 routes. Then single point cross over is used to produce children. Once the children are created each route is taken one at a time until the repeated cities in each route are replaced with the missing cities in each route. Each city is placed in one of four groups depending on whether city<4, 4<=city<8 , 8<=city<12 or 12<=city<15. Next the number of repeats and cities in each group is calculated. Based on the fact that there should be a total of four cities in each group the fitness of a city in each group is determined by the (need of its respected group)/(sum of group needs). Next the search for missing cities began. Binary values are now used to represent cities decimal equivalent. An initial population of 8 cities is randomly created. Based on the fitness of each city the roulette wheel method determines how many of each city you will have in your next population. The next population is now crossed over at random cross points (probability of each city crossing with another city =1). After crossing the cities if the fitness of a city which is calculated by the (need of its respected group)/(sum of group needs) is >=the average fitness of the cities then that city is searched for in its respected group. The cities searched for and not found are considered missing and are used to replace the repeated cities. With the missing cities found used to replace repeated cities your route will be closer to having no repeats. The previous process is continued until no repeats are found in each route and all the routes go to the 16 distinct cities.

Conclusion

I have presented a method which may be used in any genetic algorithm problem which uses a cross over that eliminates repetition and where each parent and child has to include all elements of a population. The cross over method I proposed combines grouping cross over and selection. Possible problems which may use this type of cross over are the traveling salesman and the N-queen problem.

Further Research

The method I have proposed can be used for multiple points crossover methods. It allows you to use any number of crossover points in the creation of each child. This versatility will allow for a faster convergence to the optimum solution.

References

[1] http://qspr03.tuwien.ac.at/~aschatt/info/ga/genetic.html#AlgoSelectionSubSec

[2]Homaifar,Guan, and Liepins, “Schema Analysis of the Traveling Salesman Problem Using Genetic Algortihms”, pages 533-552 in *Complex Systems Precedings* (1992)

[3]Notes taken at NCA&T in ELEN-674-01

WILSON CROSSOVER METHOD

Student: Richard Wilson
Advisor: Clinton Lee
Advisor: Abdollah Homaifar

Autonomous Control Engineering Center
Department of Electical Engineering
North Carolina A&T State University

Introduction

The idea behind GA´s is to extract optimization strategies nature uses successfully – known as Darwinian Evolution - and transform them for application in mathematical optimization theory to find the global optimum in a defined phase space.

Genetic Algorithm Steps

1.Selection

2.Mating/Crossover

3.Mutation

How Crossover Works

```
Before crossover      (1   2    3 / 4    5    6)
                      (2   4    5 / 6    3    1)
                                       ^
After crossover       (1 ®2    3    6    3®  1)
                      (2   4®  5®  4    5    6)
```

The two routes created after crossover are illegal because both go to at least one city more than once.

Example of a Two Point crossover

Davis' order crossver(OX)

OX tries to maintain the original city order in the parents in the condition of no duplication.
For the same example as above(3)

```
Tour 1          a   b | c / d   e | f / g   h
Tour 2          b   d   f / a   g   h / c   e

One of the offsprings: a   g   h   d   e   f   c   b
```

Limits on Finding Missing Cities without Genetic Algorithm

Would have to check cities randomly to see if it missing

My proposed method

Limits the numbers of cities that you search for.

Example 32 City Child 1

4 16 20 22 23 30 31 12 2 9 6 10 14 17 18
21 2 5 8 1 14 18 13 25 26 28 1 3 5 8 0 27

First Search

Group 1	Group2	Group3	Group 4
$0 \leq$ city <8	$8\leq$ city<16	$16\leq$ city <24	$24\leq$ city <32
0 1 3 5 2 4 6	9 10 14 8 13 12	16 20 22 23 17 18 21	25 26 28 30 31 27
R R R	R R	R	
Need 8 Unique	Need 8 Unique	Need 8 Unique	Need 8 Unique
Have 10	Have 8	Have 8	Have 6
Repeated 3	Repeated 2	Repeated 3	Repeated 0
1 Missing	2 Missing	1 Missing	2 Missing

f(x)=missing / total missing=fitness

Group 1	Group 2	Group 3	Group 4
f(x)=1/6	f(x)=1/3	f(x)=1/6	f(x)=1/3

Found 3 missing cities in first search (11,15,29)
Replaced 1st three repeated cities (2,14,18) with (11,15,29) respectively

Initial Pop.	Bin. To I	f(x)	f(x)/sum	Exp. Ct.	Actual Ct.
0 1 0 0 1	9	0.333	0.0833	1.333	1
1 0 1 1 0	22	0.167	0.0412	0.667	1
1 0 0 1 1	19	0.167	0.0412	0.667	0
0 1 1 0 0	12	0.333	0.0833	1.333	2
1 0 0 1 1	19	0.167	0.0412	0.667	1
0 1 1 0 1	13	0.333	0.0833	1.333	1
0 1 1 1 0	14	0.333	0.8333	1.333	2
1 0 0 1 0	18	0.167	0.0412	1.333	0
0 1 1 0 1	13	0.333	0.8333	0.667	1
1 1 1 0 1	29	0.333	0.8333	1.333	2
1 0 1 1 1	23	0.167	0.0412	0.667	0
0 0 0 1 1	3	0.167	0.0412	0.667	1
1 1 0 0 0	24	0.333	0.8333	1.333	1
0 0 1 1 1	7	0.167	0.0412	0.667	1
1 0 1 0 1	21	0.167	0.0412	0.667	1
1 1 1 1 1	31	0.333	0.8333	1.333	1

First Selection of Mating Pool

Mating Pool	After Cross	Bin to Dec	F1(x)
0 / 1 0 0 1	0 0 1 1 0	6	0.17
1 / 0 1 1 0	1 1 0 0 1	25	0.33
0 1 / 1 0 0	0 1 0 1 1	11	0.17
1 0 / 0 1 1	1 0 1 0 0	20	0.33
0 1 1/ 0 0	0 1 1 0 1	13	0.33
0 1 1/ 0 1	0 1 1 0 0	12	0.33
0 1 1 1/ 0	0 1 1 1 1	15	0.33
0 1 1 0/ 1	0 1 1 0 0	12	0.33
0 1 1/ 1 0	0 1 1 0 1	13	0.33
1 1 1/ 0 1	1 1 1 1 0	30	0.33
1/1 0 0 0	1 0 1 1 1	23	0.17
0/0 1 1 1	0 1 0 0 0	8	0.33
1 0 /1 0 1	1 0 1 1 1	23	0.17
1 1 /1 1 1	1 1 1 0 1	29	0.33
0 0 0 1 1	0 0 1 0 1	5	0.17
1 1 1 0 1	1 1 0 1 1	27	0.33

Search 9 cities (Highest Fitness)
Found 3 Missing (11,15, 29)

Search 2

Group 1	Group 2	Group 3	Group 4
[0 1 3 5 2 4 6]	[9 10 14 11 8 13 12 15]	[16 20 22 23 17 18 21]	[25 26 28 30 31 27 29]
R R	R		
Need 8 Unique	Need 8 Unique	Need 8 Unique	Need 8 Unique
Have 9	Have 9	Have 7	Have 7
Repeat 2	Repeat 1	Repeat 0	Repeat 0
Missing 1	Missing 0	Missing 1	Missing 1
F2(x)=0.333	F2(x)=0	F2(x)=0.333	F2(x)=0.333

Found 2 Missing Cities (19,24)
Replaced repeated cities (5,8) with (19,24) respectively

Initial Pop.	B. To D.	f(x)2	f(x)2/sum f	Exp. Ct.	Act. Ct
0 1 0 0 1	9	0	0	0	0
1 0 1 1 0	22	0.33	0.1	1.6	2
1 0 0 1 1	19	0.33	0.1	1.6	1
0 1 1 0 0	12	0	0	0	0
1 0 0 1 1	19	0.33	0.1	1.6	1
0 1 1 0 1	13	0	0	0	0
0 1 1 1 0	14	0	0	0	0
1 0 0 1 0	18	0.33	0.1	1.6	2
0 1 1 0 1	13	0	0	0	0
1 1 1 0 1	29	0.33	0.1	1.6	1
1 0 1 1 1	23	0.33	0.1	1.6	2
0 0 1 1 1	3	0.33	0.1	1.6	1
1 1 0 0 0	24	0.33	0.1	1.6	2
0 0 1 1 1	7	0.33	0.1	1.6	2
1 0 1 0 1	21	0.33	0.1	1.6	2
1 1 1 1 1	31	0	0	0	0

Second Selection of Mating Pool

Search 3

Group 1	Group 2	Group 3	Group 4
[0 1 3 5 2 4 6]	[9 10 14 11 8 13 12 15]	[16 20 22 23 17 18 21 19]	[24 25 26 28 30 31 27 29]
R			
Need 8 Unique	Need 8 Unique	Need 8 Unique	Need 8 Unique
Have 8	Have 8	Have 8	Have 7
Repeat 1	Repeat 0	Repeat 0	Repeat 0
Missing 1	Missing 0	Missing 0	Missing 0
F3(x)=1	F3(x)=0	F3(x)=0	F3(x)=0

Found last missing city (7)
Replaced repeated city 1 with 7.

Final Route with repeated cities (2,14,18,5,8,1) replaced by (11,15,29,19,24,7) respectively
Child 1
4 16 20 22 23 30 31 12 11 9 6 10 15 17 18
21 2 19 24 7 14 18 13 25 26 28 1 3 19 8 0 27

Use the repeated cities in Child 1 as the missing cities in child 2 and just replace them.

Initial Pop.	B to D	f3(x)	f3(x)/sum f3(x)	Exp. Ct	Act. Ct
0 1 0 0 1	9	0	0	0	0
1 0 1 1 0	22	0	0	0	0
1 0 0 1 1	19	0	0	0	0
0 1 1 0 0	12	0	0	0	0
1 0 0 1 1	19	0	0	0	0
0 1 1 0 1	13	0	0	0	0
0 1 1 1 0	14	0	0	0	0
1 0 0 1 0	18	0	0	0	0
0 1 1 0 1	13	0	0	0	0
1 1 1 0 1	29	0	0	0	0
1 0 1 1 1	23	0	0	0	0
0 0 0 1 1	3	1	0.5	8	8
1 1 0 0 0	24	0	0	0	0
0 0 1 1 1	7	1	0.5	8	8
1 0 1 0 1	21	0	0	0	0
1 1 1 1 1	31	0	0	0	0

Third Selection Pool

Mating Pool	Crossed	B to D	f3(x)
0/ 0 0 1 1	0 0 1 1 1	7	1
0/ 0 1 1 1	0 0 0 1 1	3	1
0 0/ 1 1 1	0 0 1 1 1	3	1
0 0/ 0 1 1	0 0 0 1 1	7	1
0 0 0/ 1 1	0 0 0 1 1	3	1
0 0 0/ 1 1	0 0 0 1 1	3	1
0 0 1 1/ 1	0 0 1 1 1	7	1
0 0 1 1/ 1	0 0 1 1 1	7	1
0 0 0/ 1 1	0 0 0 1 1	3	1
0 0 0/ 1 1	0 0 0 1 1	3	1
0 0/ 1 1 1	0 0 0 1 1	3	1
0 0/ 0 1 1	0 0 1 1 1	7	1
0/ 0 1 1 1	0 0 0 1 1	3	1
0/ 0 0 1 1	0 0 1 1 1	7	1
0 0 1/ 1 1	0 0 1 1 1	7	1
0 0 1/ 1 1	0 0 1 1 1	7	1

Searched 2 cities (3, 7)
Found 1 missing (7)

Final Seach Missing Cities

Mating Pool	Crossed	B TO D	F2(x)
1/ 0 1 1 0	1 0 0 1	19	0.33
1/ 0 0 1 1	1 0 1 1	22	0.33
1 0 / 1 1 0	1 0 0 1	19	0.33
1 0 / 0 1 1	1 0 1 1	22	0.33
1 0 0 /1 0	1 0 0 0	17	0.33
1 1 1/ 0 1	1 1 1 1	30	0.33
1 0 0 1/ 0	1 0 0 1	19	0.33
1 0 1 1/ 1	1 0 1 1	22	0.33
0 0 0/ 1 1	0 0 0 1	3	0.33
1 0 1/ 1 1	1 0 1 1	23	0.33
1 1/ 0 0 0	1 1 1 1	31	0.33
0 0/ 1 1 1	0 0 0 0	0	0.33
1/ 0 1 0 1	1 1 0 0	24	0.33
1/1 0 0 0	1 0 1 0	21	0.33
1 0/ 1 0 1	1 0 1 1	23	0.33
0 0/ 1 1 1	0 0 1 0	5	0.33

Repeated 19(3), 22(3), 23(2)
Search 11 Cities
Found 2 Missing (19, 24)