Project 5: Solving Multi-Objective Traveling Salesman Problem (MTSP) using Multi-Objective Evolutionary Algorithm (MOEA)

Per Magnus Veierland
permve@stud.ntnu.no

May 12, 2016

GENOTYPE

The genotype of a single individual is represented by a vector of integers describing one cyclic Traveling Salesman Problem (TSP) tour. The vector of integers holds a sequence of N integers, $s_1, s_2, \ldots, s_N$, where N is the number of cities in the TSP. The sequence is a permutation of the set $\{1, 2, \ldots, N\}$ and describes the order in which to visit each of the cities once. As an example with 5 cities: a sequence $4, 2, 5, 1, 3$ describes a tour starting at city 4, moving to city 2, moving to city 5, moving to city 1, moving to city 3, and finally moving back to starting city 4.

Excluding one city index from the genotype is an option, as one can choose to always visit the city 1 first, and city 1 must always be some place in the sequence. This should reduce the search space without adverse effects. Conclusive results on this design choice were not explored further, and a 48-city genotype ($N = 48$) was instead used.

CROSSOVER

Order Crossover (OX) (Algorithm 1) was introduced by Lawrence Davis in 1985 [1]. It is a two-parent (p_1 and p_2), two-children (c_1 and c_2) crossover operator intended to preserve the relative ordering of parent genes in offspring. Parents and offspring all share the same genome length, in this context denoted as N. Two crossover cut points are selected by first generating two uniformly random integers, *first* and *second* in the range $[0, l]$. The left cutting point is MIN(*first*, *second*), and the right cutting point is MAX(*first*, *second*). Between the cut points, c_1 is given the gene values of p_1, and c_2 is given the gene values of p_2.

For the remaining genes, the first child is assigned gene values in order from the second parent, starting from the first gene after the cutting point. Only gene values which are new to the child is assigned; gene values which the child already has are skipped. The same procedure is used to assign gene values from the first parent to the second child. As only gene values which are not already in the child genotype are assigned, and all gene positions in the children are filled, it is clear that children only end up with unique gene values.

Crossover is controlled by a crossover rate, which is a probability between $[0, 1]$ determining whether to apply the crossover operator to produce two children genotypes from two selected parent's genotypes, or to use the parent genotypes unmodified by crossover.

An infeasible child sequence in the context of the TSP genotype would be one in which a gene value (i.e. city index) is repeated. As OX only produces children where all gene values are unique, provided that the parent gene values are all unique, it is clear that infeasible children cannot be produced.

Algorithm 1 ORDER-CROSSOVER genetic operator algorithm

```
1: function ORDER-CROSSOVER(p1, p2, N)
2:     let c1 ← list of length N with nil values; let c2 ← list of length N with nil values
3:     let first ← RANDOMINTEGER(1, N); let second ← RANDOMINTEGER(1, N)
4:     let left ← MIN(first, second); let right ← MAX(first, second)
5:     for m ← left, right do
6:         c1[m] ← p1[m]; c2[m] ← p2[m]
7:     end for
8:     m ← (right + 1) mod N
9:     let i1 ← m; i2 ← m
10:    while m ≠ left do
11:        while p2[i2] ∈ c1 do
12:            i2 ← (i2 + 1) mod N
13:        end while
14:        while p1[i1] ∈ c2 do
15:            i1 ← (i1 + 1) mod N
16:        end while
17:        c1[m] ← p2[i2]; c2[m] ← p1[i1]
18:    end while
19:    return c1, c2
20: end function
```

MUTATION

The mutation operator is used to maintain genetic diversity in the population. For this problem, a swapping mutation operator was chosen (see Algorithm 2). For a given genotype sequence, $s_1, s_2, \ldots, s_N$, two random integer indexes a and b are randomly drawn from a uniform distribution $[1, N]$. The genotype sequence is mutated by swapping the values of genes s_a and s_b.

Mutation is controlled by a mutation rate, which is a probability between $[0, 1]$ determining whether to mutate an individual's genotype.

An infeasible child sequence in the context of the TSP genotype would be one in which a gene value (i.e. city index) is repeated. As the SWAP-MUTATION algorithm only swaps two values in a genotype sequence, it will never remove or introduce new values. Therefore, if all values in a sequence is unique before mutating the sequence, it will remain unique after mutating the sequence.

Algorithm 2 SWAP-MUTATION genetic operator algorithm

```
1: function SWAP-MUTATION(s, N)
2:     let a ← RANDOMINTEGER(1, N)
3:     let b ← RANDOMINTEGER(1, N)
4:     let t ← s[a]
5:     s[a] ← s[b]
6:     s[b] ← t
7: end function
```

SELECTION STRATEGY

The MOEA implementation uses a tournament selection strategy for parent selection. To select a single parent, a random sample of k individuals are drawn from the population P_t. With a probability of ϵ, a random individual is selected from the group of k individuals, and with a probability of $1 - \epsilon$; the lowest ranked individual in the group of k individuals is selected according to the *crowded-comparison operator* $\prec_n$ which is described in NSGA-II [2]. This procedure is performed twice to select two parents for each mating which results in two offspring. An ϵ value of 0.1 was always used.

Parameter set	Population	Generations	Crossover rate	Mutation rate	Tournament group size	Best distance	Worst distance	Best cost	Worst cost	Non-dominated front size
A	100	200	0.8	0.005	20	41525	95445	477	1165	100
B	500	200	1.0	0.005	25	36365	128950	366	1479	500
C	1000	200	1.0	0.05	200	34869	135113	324	1693	1000

Table 1: Summary of the top three parameter combinations.

PARAMETER SELECTION

To evaluate patterns in the effects of parameter selection a parameter search was conducted. For each combination of a) population sizes (50, 100, 500, 1000), b) group size ratios (0.05, 0.1, 0.2), c) crossover rates (0, 0.2, 0.4, 0.6, 0.8, 1.0), and d) mutation rates (0.001, 0.005, 0.01, 0.05, 0.1, 0.5); five evolutionary runs were evaluated, for a total of 2160 runs (see Appendix). For each data point the results from five runs are averaged. By combining all the generated solutions, a "faux Pareto front" was established and used to calculate estimated convergence and diversity metrics for every non-dominated front. Through interpreting the resulting data it can be concluded that: 1) lower crossover rates and high (but not too high!) mutation rates result in greater diversity, 2) some crossover is necessary for convergence, 3) high crossover rates and low mutation rates result in better extreme solutions, and 4) greater population sizes result in better extreme solutions for the same number of generations.

To estimate the extremes in the true Pareto front, a 1-tree lower bound [3] was calculated as 28510 for distance and 134 for cost. As the "1-tree lower bound" is usually 10% below the optimal solution, the optimal distance solution is estimated to be ≈ 31678, and the optimal cost solution is estimated to be ≈ 149.

COMMENTS

To eliminate a bias introduced by the `crowding-distance-assignment` function, it was found necessary to shuffle the last front added to the next population. Otherwise, if only part of the last front is added to the next population, having the front sorted by one of the objective functions creates a significant bias towards the lower end of the last objective function sorted for. This in turn can lead to a clustering of solutions on one of the ends of the population front.

As the true Pareto front is unknown to the implementation for the problem given, $f_m^{\max}$ and $f_m^{\min}$ are estimated for each objective function m by tracking the maximum and minimum values encountered throughout an evolutionary run for each objective.

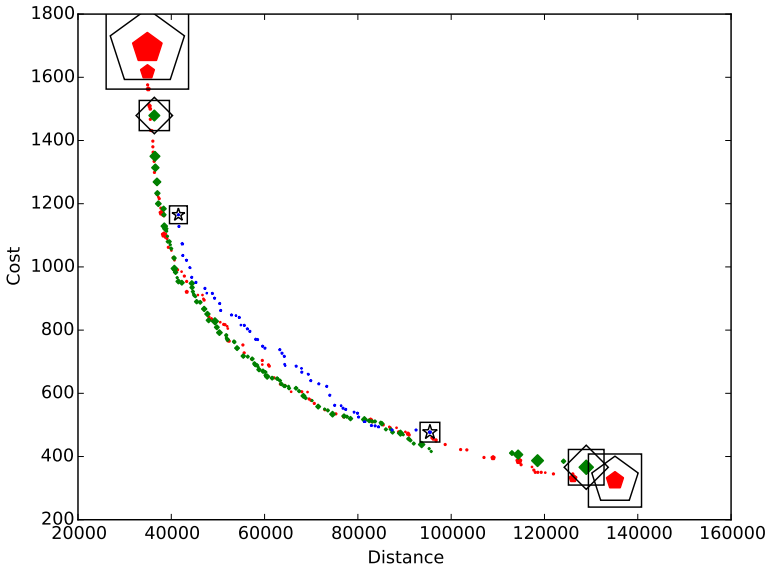

Figure 1: Non-dominated fronts for parameter set A (blue stars), parameter set B (green diamonds), parameter set C (red pentagons). Shape sizes indicates clustered solutions. Shape outlines indicates best solution for an objective while square outlines indicates worst solution for an objective.

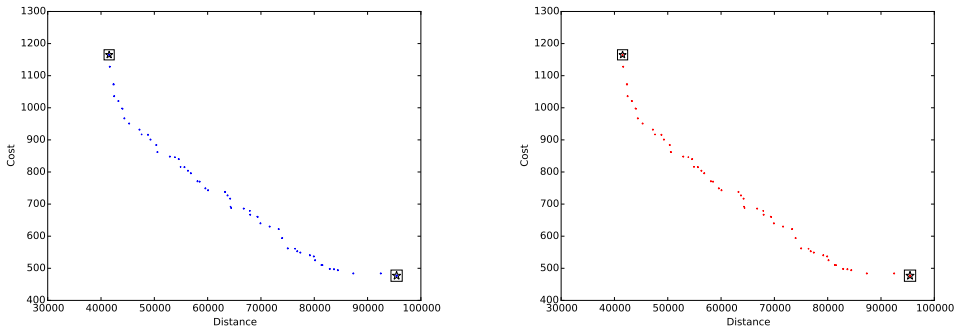

Figure 2: Non-dominated front (*left*) and full population (*right*) for parameter set A.

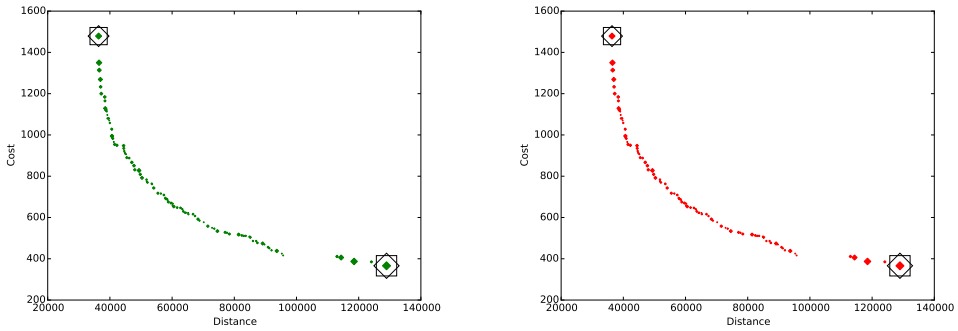

Figure 3: Non-dominated front (*left*) and full population (*right*) for parameter set B.

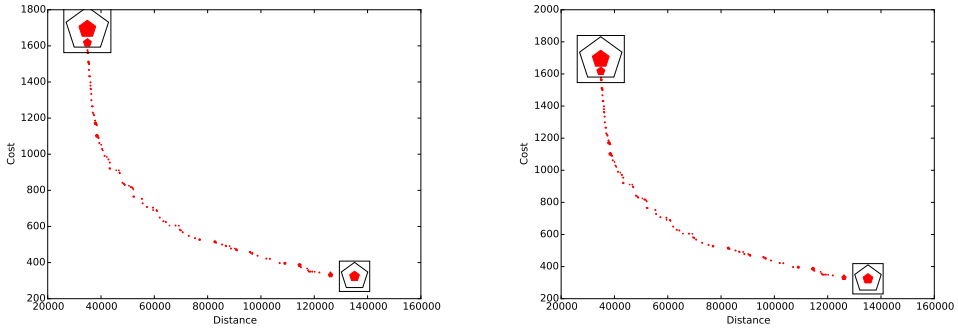

Figure 4: Non-dominated front (*left*) and full population (*right*) for parameter set C.

REFERENCES

[1] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164, 1985.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[3] Luis Goddyn. TSP Lower Bounds. `http://people.math.sfu.ca/~goddyn/Courses/800/Resources/Algorithms/TSPlowerbounds.pdf`.

APPENDIX: PARAMETER SEARCH

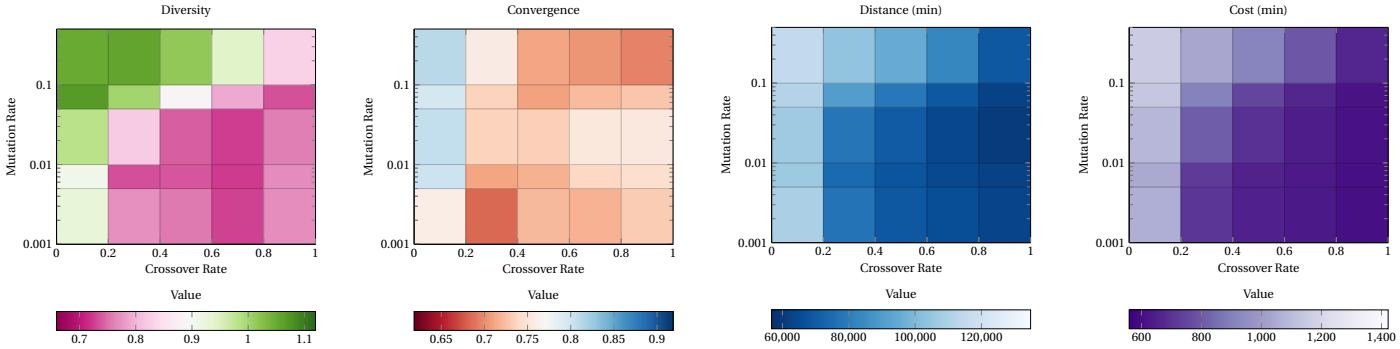

Figure 1: Population: 50 – Generations: 200 – Group Ratio: 0.05

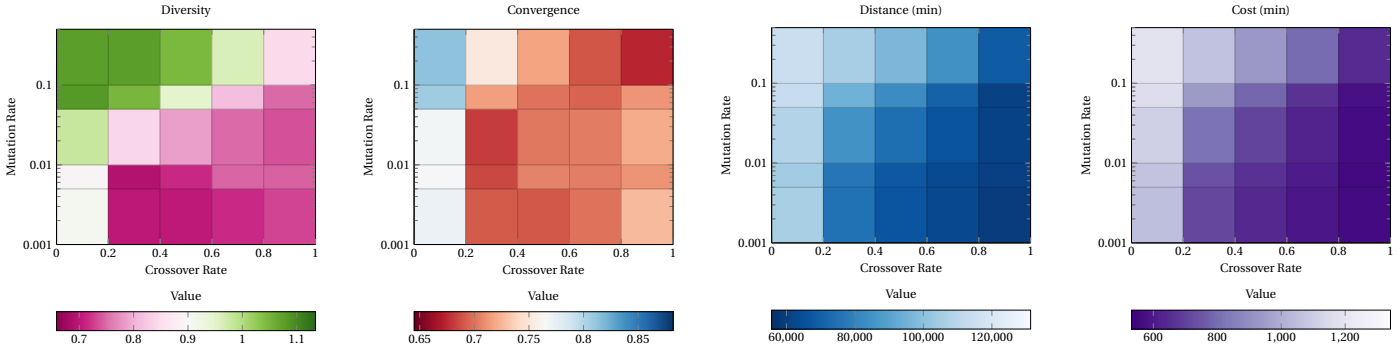

Figure 2: Population: 50 – Generations: 200 – Group Ratio: 0.1

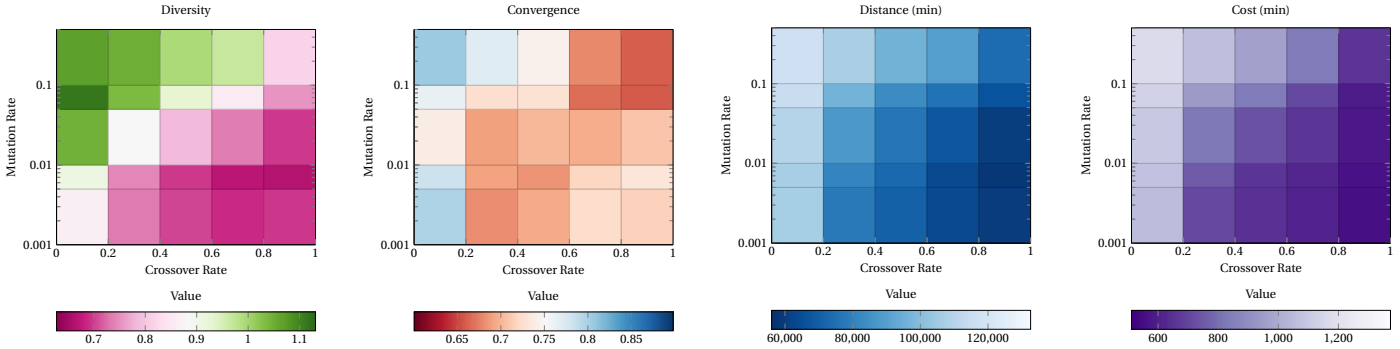

Figure 3: Population: 50 – Generations: 200 – Group Ratio: 0.2

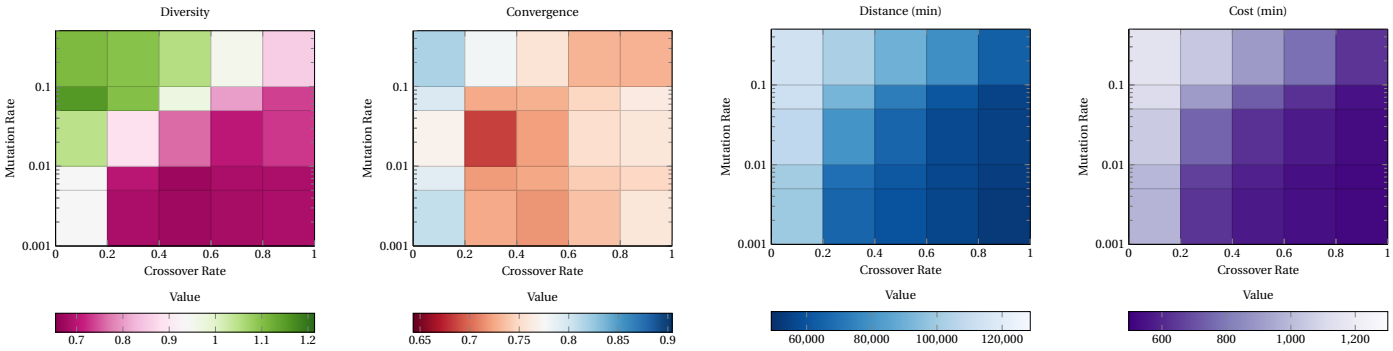

Figure 4: Population: 100 – Generations: 200 – Group Ratio: 0.05

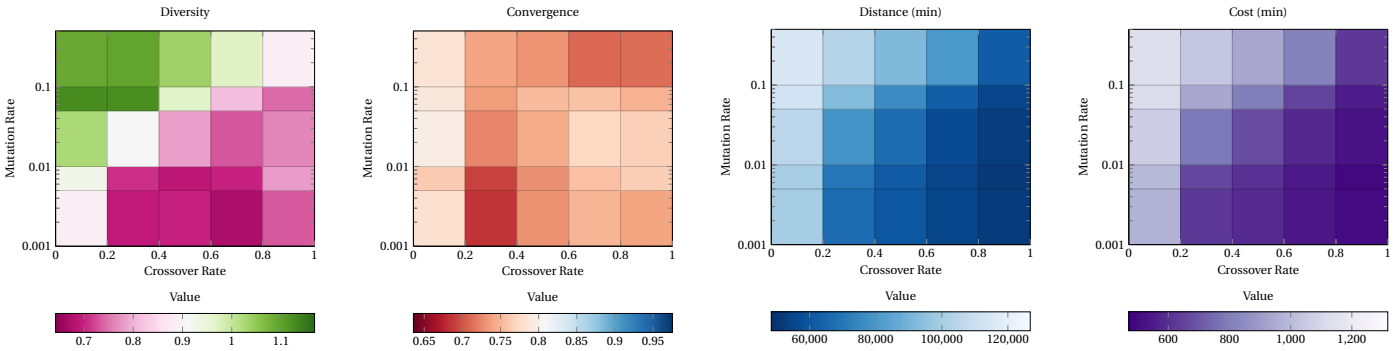

Figure 5: Population: 100 – Generations: 200 – Group Ratio: 0.1

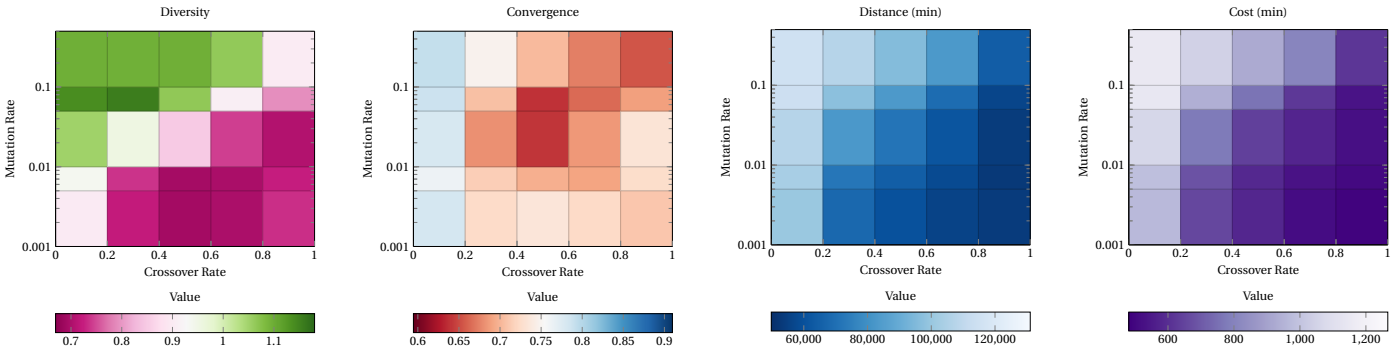

Figure 6: Population: 100 – Generations: 200 – Group Ratio: 0.2

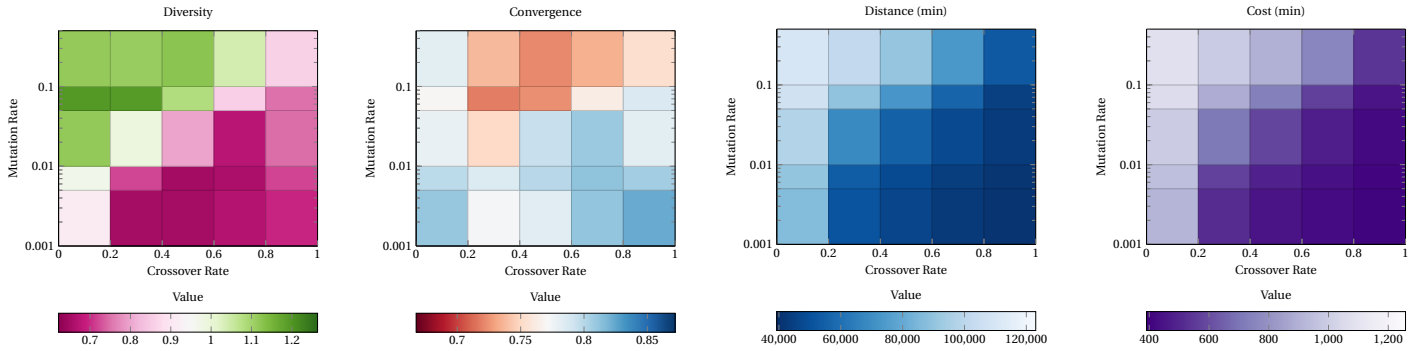

Figure 7: Population: 500 – Generations: 200 – Group Ratio: 0.05

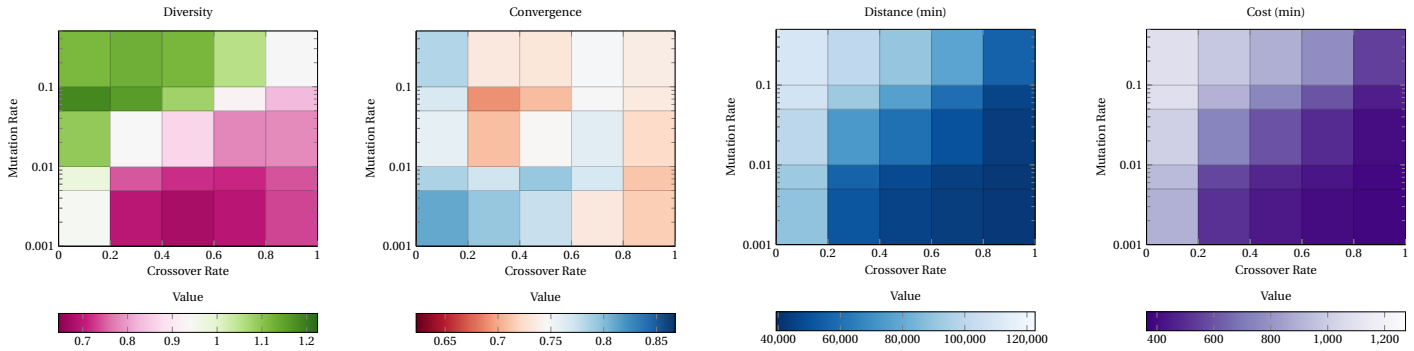

Figure 8: Population: 500 – Generations: 200 – Group Ratio: 0.1

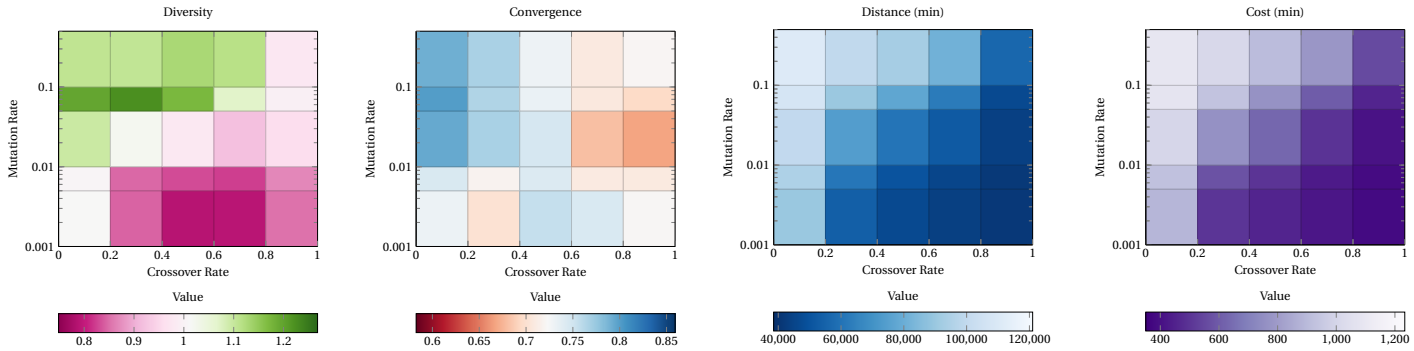

Figure 9: Population: 500 – Generations: 200 – Group Ratio: 0.2

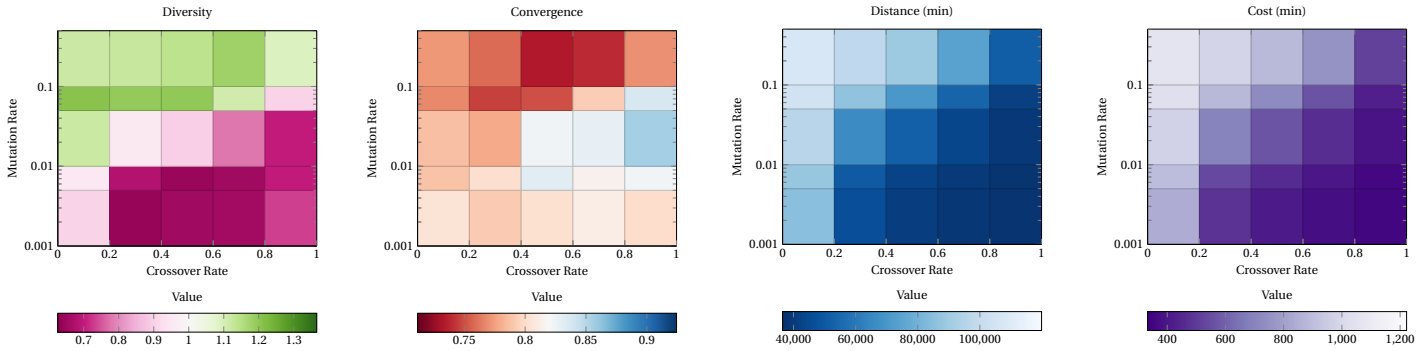

Figure 10: Population: 1000 – Generations: 200 – Group Ratio: 0.05

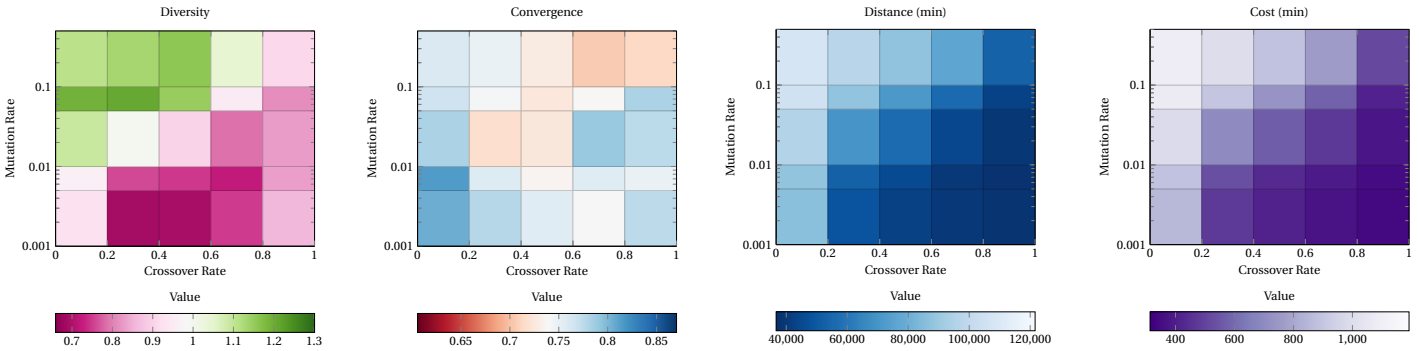

Figure 11: Population: 1000 – Generations: 200 – Group Ratio: 0.1

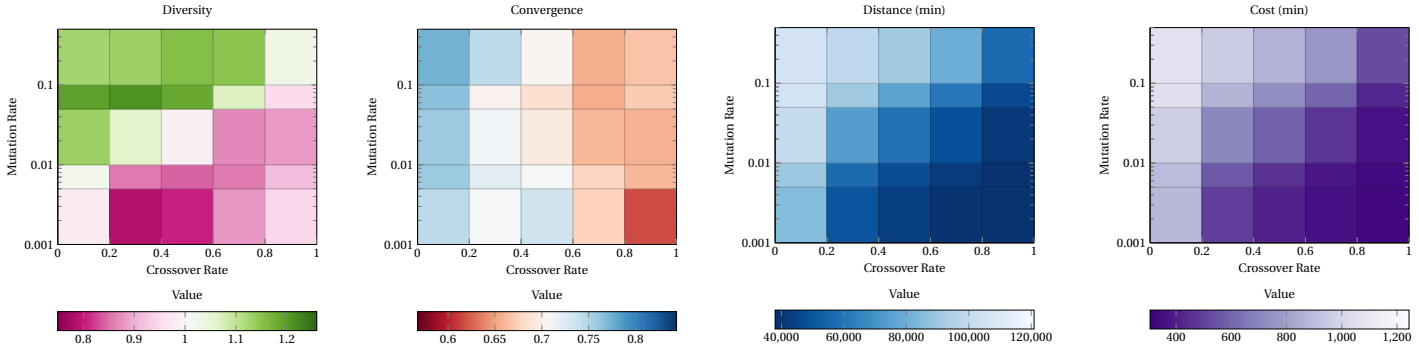

Figure 12: Population: 1000 – Generations: 200 – Group Ratio: 0.2