Lab Exercise 3, Simulation

Joel Bångdal
Viktor Claesson

Kurs
EITN95
Lund Universitet

May 19, 2020

Exercise 1

Part a

city	genetic	tabu	simulated annealing
loadatt48	33723	38071	36583
loadbays29	9074	9791	9197
loadeil101	677	731	1918
loadeil535	3977	2500	19157
loadgr96	527	575	1832
loadpcb442	94208	61223	162490
loadpr76	120412	127792	151812
loadst70	696	778	1587

Table 1: Total-distance with all default settings, across all cities.

There is no superior method, it's all based on number of points and their layout. But the genetic one seem to be the one that performs the best across most of the layouts.

Part b

run	genetic	tabu	simulated annealing
1	690	731	1777
2	674	731	1969
3	677	731	1935
4	680	731	2184
5	675	731	2107
6	665	731	2176
7	680	731	1792
8	673	731	1846
9	674	731	1976
10	676	731	2022

Table 2: The total distance of ten runs with the default values on 101 cities

SA is by far the worst, having a worse result and fluctuating a total of 407 between runs while TA is stable for all but not the most optimised. GA is the superior choice since it produces the best result while only fluctuating 35 points.

GA: small fluctuation between runs since we may hit local maximums that is not the best solution.

TS: We can assume that there is a very deep local minimum at 731 since we always get there in the end and tabu-search would have to increase the amount of backtracking to get a better result.

SA: The reason there is such a great discrepancy is because the cooldown-period is to small, and the runtime is long for higher number of iterations. Since we don't let it cool down slowly over a longer period of time the result is not going to be very accurate.

Part c

Inital temp.	Cooling rate	Iterations	Iterations/temp	Result
1500	0.970	2000	10	1654
2000	0.970	2000	10	2070
2500	0.970	2000	10	1963
3000	0.970	2000	10	2239
2000	0.960	2000	10	948
2000	0.970	2000	10	2040
2000	0.980	2000	10	3130
2000	0.990	2000	10	3067
2000	0.970	1500	10	3071
2000	0.970	2000	10	1984
2000	0.970	3000	10	825
2000	0.970	4000	10	756
2000	0.970	2000	5	844
2000	0.970	2000	10	1878
2000	0.970	2000	20	2998
2000	0.970	2000	30	3464

Table 3: SA: 101

The data itself speaks pretty well for how the result is effected by each individual change, the takeaway is that a balance between these variables is crucial.

Tabu tenure	Number of iterations	Result
10	10000	61223
20	10000	61223
30	10000	61223
40	10000	61223
20	5000	61223
20	10000	61223
20	20000	61223
20	50000	61223

Table 4: TA: 442

No matter what variables we changed the result stayed the same, even with different cities and a mix och all-low, all-high and one-high one-low variable.

Population size	Number of iterations	Last Iteration	Result
50	10000	8410	674
100	10000	9681	670
200	10000	9894	667
400	10000	8342	664
600	10000	4835	675
800	10000	5736	670
100	5000	4919	697
100	10000	9941	653
100	20000	19117	663
100	50000	38725	661
100	100000	99888	670

Table 5: GA

There is no hard, correlation between any of the inputs and the result. It all seems to depend on if you hit a bad local maximum or not in the algorithm.

Part d

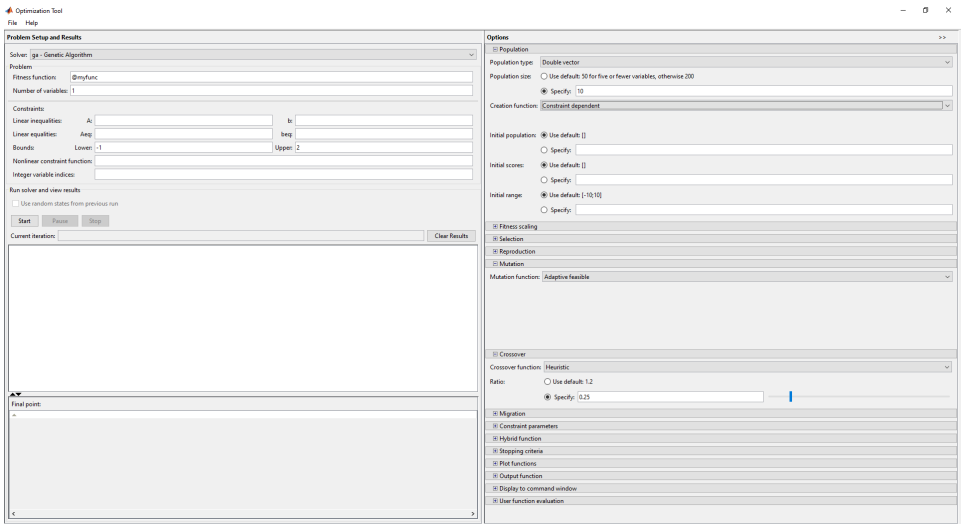
This is the crossover code added to the Genetic algorithm, and is placed after the Genetic Algorithm Operators but before pop is set to newPop.

The make_child function is defined within the tsp_ga function and used by the crossover to mate two parents. It uses the algorithm described in lecture H3.

```
randomOrder = randperm(popSize);
for p = 2:2:popSize
    if(rand <= 0.05)
        parent1 = pop(randomOrder(p-1), :);
        parent2 = pop(randomOrder(p), :);
        newPop(randomOrder(p-1), :) = make_child(
            parent1, parent2);
        newPop(randomOrder(p), :) = make_child(parent2
            , parent1);
    end
end
pop = newPop;
```

```
function child = make_child(parent1, parent2)
    [~, N] = size(parent1);
    subset_points = sort(ceil(N*rand(1,2)));
    subset = parent1(subset_points(1):subset_points(2)
        );
    child = zeros(size(parent1));
    child(subset_points(1):subset_points(2)) = subset;
    j = 1; %Last searched node
    for i = 1:subset_points(1)-1
        while(ismember(parent2(j), subset))
            j = j + 1;
        end
        child(i) = parent2(j);
        j = j + 1;
    end
    for i = subset_points(2)+1:n
        while(ismember(parent2(j), subset))
            j = j + 1;
        end
        child(i) = parent2(j);
        j = j + 1;
    end
end
```

Exercise 2




```
function [x,fval,exitflag,output,population,score] =
    ga_2(nvars,lb,ub,PopulationSize_Data,
    CrossoverFraction_Data)

    function [z] = myfunc(x)
        z = x*sin(x*pi*10)+1.0;
    end

    options = optimoptions('ga');
    options = optimoptions(options,'PopulationSize',
        PopulationSize\textunderscore Data);
    options = optimoptions(options,'CrossoverFraction'
        , CrossoverFraction\textunderscore Data);
    options = optimoptions(options,'MutationFcn', {
        @mutationadaptfeasible 0.04 });
    options = optimoptions(options,'Display', 'off');
    [x,fval,exitflag,output,population,score] = ga(
        @myfunc,nvars,[],[],[],[],lb,ub,[],[],options);
end
```

run with ga_2(1,-1,2,10,0.25)