Project Report
Genetic Algorithms and Evolutionary Computing [H02D1A]

Jasper Bau, Daan Seynaeve

January 5, 2016

1 Parameter Experiments

```
rondrit16.tsp
--
default settings:
3.5837

Cannot be optimal. Visibile inneffiencies in the tour. Can be optimized by changing only 2 edges. (s

run 1
Best solution was already there 50 generations ago

run2
not the case


minder generaties (100 default):
50: 3.81, 4.10, 4.01, 3.56, 3.81
(meer crossing edges)
(greater tour length)

25: 3.94, 4.17, 3.57, 4.2, 4.3


minder individuals (50 default):
25: 4.07, 3.55, 3.58, 3.64, 3.07
(less diversity)

meer indivduals:
200: 3.49, 3.45, 3.35, 3.36, 3.41
(consistent result, no crossing edges)

100 individuals, 50 gen:
```

```
3.66, 3.67, 3.50, 3.51, 3.54


No elites:
(No convergence, oscillating, random behaviour)
bad results
more generations --> no influence
more individuals --> less extreme oscillation. Average fitness levels out. Seems to be stuck at subo

More individuals warrants more elite?

High amount of elites -> stuck at specific solutions for a long time. Population converges to a sing

Enabling loop detection -> much better performance. Faster saturation + convergence.

Prob. for mutation or crossover must be high enough for anything to happen...

If no mutation -> algo can become 'stuck'.

no mutation + low crossover prob -> algo comes to an early stop.

crossover increasingly important for larger problem instances?
```

2 Path Representation

Some operators: A.2 and A.1

3 Optional Task

4 Benchmark Performance

Appendix A Code

A.1 scx.m

```
% Sequential Constructive crossover for TSP (Zakir H. Ahmed, 2010)
% this crossover assumes that the path representation is used to represent
% TSP tours
%
% Daan Seynaeve 2015
%
%
% Syntax:  NewChrom = scx(OldChrom,Cost)
%
% Input parameters:
%    OldChrom  - Matrix containing the chromosomes of the old
%                population. Each line corresponds to one individual
%                (in any form, not necessarily real values).
%
%    Cost      - Cost matrix. Cost(i,j) corresponds to the cost between
%                the node i and the node j
%
% Output parameter:
%    NewChrom  - Matrix containing the chromosomes of the population
%                after mating, ready to be mutated and/or evaluated,
%                in the same format as OldChrom.
function NewChrom = scx(oldChrom,Cost,x_probability)
    if nargin < 3
        x_probability = 1;
    end
    if rand<x_probability
        n = size(oldChrom,2);
        child1 = zeros(1,n);
        legit = ones(1,n);

        parent1 = oldChrom(1,:);
        parent2 = oldChrom(2,:);

        nci = 1;
        child1(1) = 1;
        legit(1) = 0;
        while nci < n
            p = child1(nci);
            nci = nci + 1;

            alfa = 0;
            beta = 0;

            for i = 1:n-1
                if parent1(i) == p && legit(parent1(i+1))
                    alfa = parent1(i + 1);
                end
                if parent2(i) == p && legit(parent2(i+1))
                    beta = parent2(i + 1);
                end
            end

            if alfa == 0 || beta == 0
                % find a legit k
                k = 2;
                while not(legit(k))
                    k = k + 1;
                end

                if beta == 0 && alfa == 0
                    beta = k;
                    alfa = k;
                elseif beta == 0
                    beta = k;
                else
                    alfa = k;
                end
            end
            if Cost(p,alfa) < Cost(p,beta)
                child1(nci) = alfa;
            else
                child1(nci) = beta;
            end
            legit(child1(nci)) = 0;
        end
        NewChrom = [parent1; child1];
    else
        NewChrom = oldChrom;
    end

end
```

A.2 cycle_cross.m

```
% Cycle crossover for the TSP problem
% Crossover operator for the path representation

function NewChrom = cycle_cross(OldChrom, x_probability)
    if nargin<2
        x_probability = 1;
    end
    if rand<x_probability
        parent1 = OldChrom(1,:);
        parent2 = OldChrom(2,:);

        % construct a cycle and use it as offspring
        start_cycle = parent1(1);
        next_in_cycle = parent2(1);
        child1 = [start_cycle zeros(1,length(parent1)-1)];
        child2 = [next_in_cycle zeros(1,length(parent2)-1)];

        while next_in_cycle ~= parent1(1)
            index_other_parent = find(parent1 == next_in_cycle);
            child1(index_other_parent) = next_in_cycle;
            next_in_cycle = parent2(index_other_parent);
            child2(index_other_parent) = next_in_cycle;
        end

        % exchange the elements that are not copied yet
        positions_zero = find(child1==0);
        for i=positions_zero
            child1(i) = parent2(i);
            child2(i) = parent1(i);
        end

        NewChrom(1,:) = child1;
        NewChrom(2,:) = child2;
    else
        NewChrom = OldChrom;
    end
end
```