Institut for Matematik & Datalogi
Syddansk Universitet

5. oktober 2005
JBJ

DM63 Meta-heuristics — Ugeseddel 5

Lecture on October 6, 2005

I lecture on genetic algorithms. Notes pages 29-35 and 134-162 (last part will only be covered partly at the lecture).

Plan for October 13, 2005

I will lecture on the noising method and guided local search.

Exercises:

1. Implement a raw genetic algorithm for the graph partitioning problem together with procedures for calculating each of the following
   (a) An initial population of size K. This could consist of simply K random solutions (partitions) or it could contain some good solutions found using one of the heuristics you have worked with such as Decent, SA, TS etc.
   (b) An algorithm for evaluating the current population. This must include a method for calculating the fitness of a solution.
   (c) An algorithm for selecting a basis population based on fitness values (If you choose to use the ranking method (page 169 in Reeves) the fitness corresponds to using this ranking as the probability distribution based on which you select individuals for the basis population). This selection procedure for the next popolation must satisfy the rule that a selected solution is transfered directly to the next generation with probability p_d and is crossed with another selected individual with probability $1 - p_d$ after which the resulting solutions are transfered to the next generation .
   (d) A method for crossing two individuals.
   (e) A method for mutating an individual.
   (f) A method for graphically displaying the objective function of intermediate solutions (as you did for SA and TS, use Gnu plot for instance).
2. Experiment on various input graphs in order to investigate the following
   (a) How much does the initial population affect the quality of the solution? Try e.g. to start with K almost identical solutions. Try also to start with K good solutions found by other heuristics. Does this lead to better solutions than starting from K random solutions?
   (b) Try to determine the number of populations the algorithm needs to investigate before the stopping criterion is reached. Is there any visible convergence?
   (c) Try to find a good relation between the size K of the population and the quality of the best solution after a fixed (large ??) number of iterations.
   (d) Experiment with various cross over strategies including those which we discussed at the lectures.
   (e) Try to allow unbalanced partitions (using a penalty factor). Does this lead to better solutions?
   (f) Experiment with various methods for making mutations. Should one mutate a lot or only seldomly?
   (g) Try varying p_d and observe the effect of this. What is the best value for p_d and is the a safe range where the algorithm behaves nicely?

Various schemes for performing a cross over: To supplement your own ideas for cross over operators you can try the following. Here we assume that solutions are represented by a bit vector where $s[i] = 1]$ means that i is in the set X and $s[i] = 0$ means that i is in the set Y (when we denote the partition X, Y). Furthermore we assume that s'' is returned as the result of the crossing.

1. Choose a random position i. For $j := 1$ to i let $s''[j] := s[j]$. For $j > i$ we assign $s''[j]$ the value of $s[j]$ and increase j to $j + 1$ until we either have $j > n$ or we have $n/2$ ones or zeroes in s''. In the last case we fill up with zeroes and ones respectively.

2. Choose random i, j such that $1 \leq i \leq j \leq n$. If the number of ones in $s[i..j]$ equals the number of ones in $s'[i..j]$ then let $s'' := s[1..i-1]s'[i..j]s[j+1..n]$. Otherwise try a new pair i, j. After r (a parameter) unsuccessful attempts we let $s'' := s$.

3. The variant of 2. where at each failed attempt try to increase j until there are equally many ones in $s[i..j]$ and $s'[i.j]$ and then perform the swap.