Verslag

Bert Mortier, Toon Nolten

1 Introduction

The goal of this project is to find a good genetic algorithm (GA) to solve the traveling salesman problem (TSP). This means we have to find good crossover operators and good mutation operators, but also tune the several parameters involved. In order to do this we first of all implemented a set of additional crossover and mutation operators.

The main problem is finding out which sets of operators and parameters perform well. An option would be to test many combinations and evaluate which ones give a good performance. However, even for a very small TSP, the solution by a genetic algorithm takes a significant amount of time. So to test thousands of parametersets would take a prohibitive amount of time.

There are several ways around this. A first option would be to try to optimize each of the parameters seperately. This ignores a very important aspect in our opinion, namely the interplay between different operators. It could for instance be that a certain mutator operator only performs well when it is combined with a specific crossover operator.

A second option would be to start with a very crude mesh on the parameterspace and find new, narrower ranges for each of the parameters after each iteration by interpreting the performance of each of the parametersets. This would be a good option if it weren't for the very long runtime each evaluation of the parametersets would take. Each time an entire TSP has to be solved and that soon takes close to a minute of time. This means we would need a very crude mesh, which in turn implies several iterations before we found the parametervalues with a sufficient precision. Besides the fact that this means we would have a very manual approach, it also means we need alot of human interpretations. We are hesitant to include many of those since it would mean we introduce many of our prejudices about operators, which might not be valid. This would damage one of the strengths of a genetic algorithm to find unexpected solutions. An example of this is the earlier mentioned posibility of crossover and mutation operators that only perform well in combination for which it might indeed be hard to foresee in advance.

2 Solution: meta-GA

Our solution to this problem is a more subtle one. We scan the entire parameterspace by using a genetic algorithm, which we dub the meta-GA. This is an appropriate choice since want to find the optimum in a very complex search space with (probably) many local minima.

As a representation we have used a string which contains integers, real-valued numbers and also function handles. For different tests we have used different strings, but that is more thoroughly discussed in the section with results. Our most straight forward experiments used, among others, the amount of individuals, the maximum number of generations, the elitsm percentage, the crossover probability, the crossover operator and the mutation operator as variables in our representation.

To perform crossover and mutation for the meta-GA we have used fairly simple operators, namely single-point crossover and one-point mutation. In a representation as simple as ours this seems sufficient. We did make sure to put the crossover probability and crossover operator close together and similarly for the mutation probability and operator. This way the crossover has a large chance of moving the entire ̀crossover blocḱfrom one parent. It is of course important to emphasize that the parameters and operators of our meta-GA don't have to be optimal, in contrast to the ones for the normal GA. The reason is that the GA parameters are our goal, whereas the meta-GA only constitutes the means to find them. For instance, if our desired solution would be the optimal path for a specific TSP, then the parameters of the GA wouldn't be that crucial either.

A possibility for a fitness function of the meta-GA would be to take the best pathlength as found by a GA with the parameters as provided by the individual of the meta-GA. However, one problem that might occur here is the tendency to increase the number of individuals and of the maximal amount of generations. That's why we have added a sort of regularization term to our fitness function.

3 Regularization

A first option for the regularization term is to simply include a term that increases linearily with the measured time of the GA. Matlab provides the function `tic toc` to do this, but it is highly variable, so we decided not to use this. Instead we have determined for each of the operators a relative computational cost and we assume the total computational cost scales with the number of individuals (N_{ind}) and the maximum number of generations ($maxgen$). This gives us the fitness function shown as equation 1, where $cost_{operator}$ is the cost for a certain operator and $Pr_{operator}$ is the probability of using this operator. α is a parameter we use to decide how heavily we weigh this regularization.

$$f = shortestPathlength + \alpha \cdot maxgen \cdot N_{ind} \cdot \sum_{operator} Pr_{operator} \cdot cost_{operator} \quad (1)$$

There are several possible heuristics that might give us a good value for this α, but we have initially just used a fairly small value which we will adapt if we see a tendency to very small or very big population sizes. To determine the $cost_{operator}$ values, we have taken the mean of many `tic toc` experiments, untill we have a standard deviation of less than 10%. One remark that has to be made here is that we now introduce a tendency towards operators that are well suited for Matlab, so our solution will be tuned towards Matlab. However, we feel that this is acceptable, since we are working in a Matlab environment and since the difference in cost for each of the operators is a relevant design parameter.

4 Results

5 Expansion: adaptivity

Our way of working is particulary well designed to include adaptivity. We simply have to introduce some extra parameters in the meta-GA.

6 Conclusion