Aarhus Universitet

ERTS - Group 2

Rapport

Assignment 2

Gruppemedlemmer:	**Studienumre:**
Daniel Tøttrup	201509520
Mathias Lønborg Friis	201505665
Stinus Lykke Skovgaard	201401682

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

18. december 2019

Kapitel 1

INTRODUCTION

This report describes a project for using HW/SW Co-design in designing and modeling a “Route Optimization using Genetic Search Algorithm” (ROGSAnne). The project is about defining a methodology and using it to describe a model of a system. After this some of the system will be designed for the software part, and an IP core will be modeled and tested using the Vivado tool chain.

1.1 The problem

The purpose of the system is to find an optimum route between a series of points, minimizing the total traveled distance. This problem is commonly known as the “Traveling Salesperson Problem” [ref: wiki?].

The optimized route plan is intended to be used within a Drone Delivery System, in which a drone has to deliver an amount of packages (e.g. 100, it’s a very large drone) to different locations. It is imagined that the ROGSAnne-system would be implemented on the server-side of a system, allowing for a route plan to be calculated, provided the locations of the points to visit for a drone.

The problem is that the number of possible candidate solutions raise exponentially (maybe do some calculations?) with the number of points to visit. Instead of calculating the traveled distance of each possible route, an optimization algorithm can be used to find the best route. In this case, the meta-heuristic optimization algorithm “Genetic Algorithm” is used.

While using an optimization algorithm can speed up the process of finding the optimum route, it is still a computationally heavy task. Thus, it is of interest to speed up the process using hardware acceleration, allowing more drones to utilize the same server.

As the production of each new candidate solution in a new “generation” is independent of each other, it is assumed that the process can be parallelized, greatly improving calculation speed.

In a real system, the coordinates would be provided by the Drone Delivery System. In this project, the coordinates will be read from a file on an SD card.

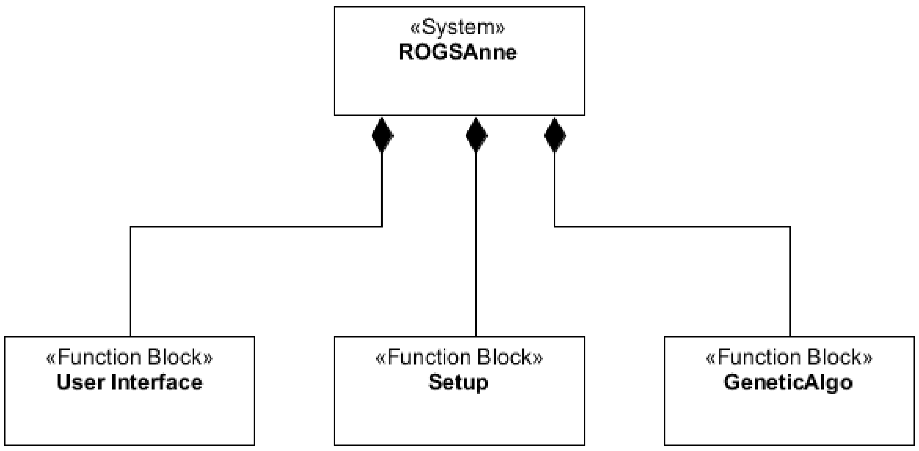
Kapitel 2

METHODOLOGY

The following chapter will describe the groups methodology and show multiple diagrams that show the behaviour of the system. These diagrams will be the following.

- **Block Definition Diagram(BDD):** To provide an overview of the hardware structure.
- **Internal Block Diagram (IBD):** To provide an overview of the connection between IP-blocks.
- **Class Diagram:** To describe the logical partitions of the implemented software, and their dependencies and associations.
- **Activity Diagram:** To provide an overview of the overall system flow.

The block definition diagram show the overall context of the system and what the it consists of.

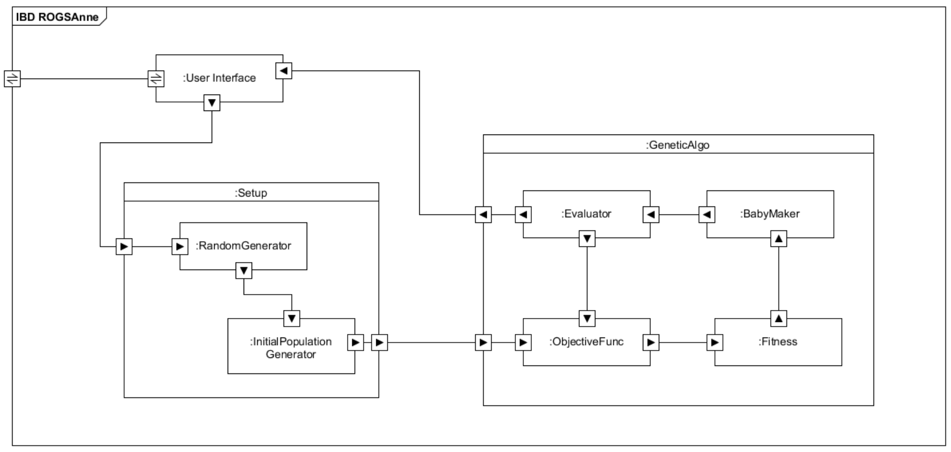


Figur 2.1: Block definition diagram of ROGSAnne

The system consists of three function blocks. These blocks encapsulates some sort of functionality.

- **User Interface** handles the interface between user and system. This is done through a console.
- **Setup** handles the initial creation of a population for the genetic algorithm.
- **GeneticAlgo** will handle the optimization of the route with a genetic search algorithm.

An internat block diagram is made based on the BDD below.



Figur 2.2: Internal block definition diagram of ROGSAnne

Kapitel 3

REQUIREMENTS

Kapitel 4

DESIGN

4.1 Design Patterns

4.1.1 State Machine Pattern

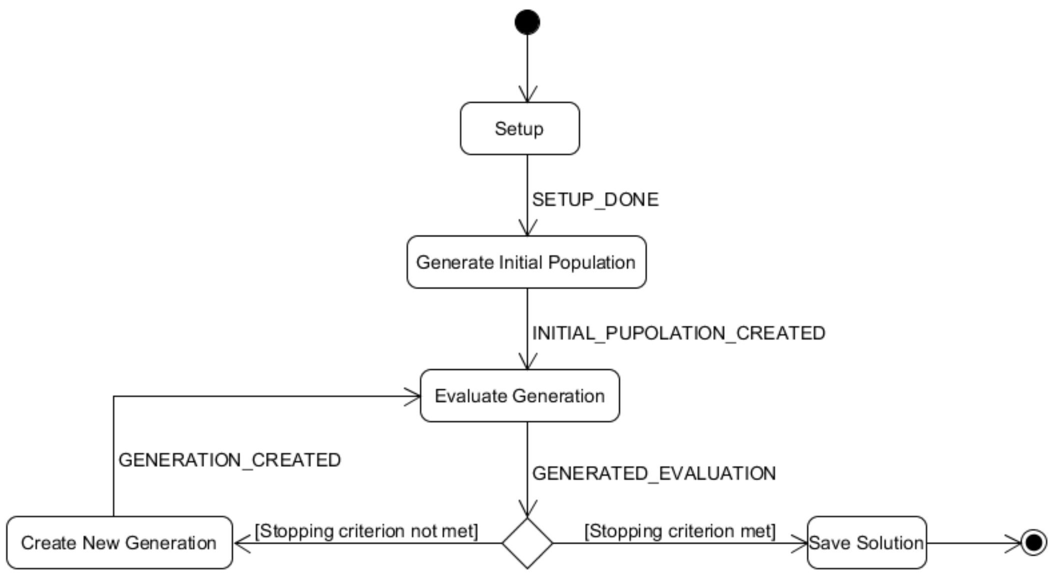
The state machine pattern has been used to develop the system. This is partly in order to structure the general flow of the algorithm, and partly in order to support concurrent development of the algorithm, as different parts of the algorithm can be implemented simultaneously as different states.

Figure 4.1 shows the state machine diagram for system. It can be seen that the general flow of the system is as follows:

1. Set up system.
2. Create initial population.
3. Evaluate generation.
4. If stopping criterion is met, save population and stop. Else- create new generation, then return to step 3.

4.1.2 Command Pattern

The command pattern will be utilized in the creation of new generations. This is done to add the possibility of parallellizing the process of creating samples in the new generation. By creating a command queue and letting different instances of the class that creates samples *(creators)* execute the commands, it is possible to "order"the creation of a certain amount of new samples, and then let the execution be up to the scheduler. This way, design space exploration can be carried out, with evaluation of execution speed versus resources used.



Figur 4.1: State machine diagram for the developed system.

Kapitel 5

THEORY

This section will describe the theory behind the problem worked with in this project.

5.1 Traveling Salesman Problem

The problem is basically a well-known optimization problem called the Traveling Salesman Problem[1]. The problem consists of creating the shortest route between a set of points, with all points being visited exactly once.

Given points with 2 coordinates, and assuming "as-the-crow-flies-movement, the distance between two points is merely calculated as the euclidean distance between the points. For two points A and B, the distance is calculated as follows:

$$dist(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2} \tag{5.1}$$

where x_A and x_B are the x-coordinates of points A and B respectively, etc...

The total distance of the route is then calculated as:

$$dist_{total}(\mathbf{points}) = \sum_{i=0}^{N-1} (dist(\mathbf{points}(N), \mathbf{points}(N+1))) \tag{5.2}$$

where **points** is a vector of points, in the order they are to be visited and **points**(N) is the Nth entry in the vector.

The problem is that the number of possible solutions increases with the number of points at a rate of:

$$nPossibleSolutions = !nPoints \tag{5.3}$$

As the number of possible solutions is equal to the factorial of the number of points, this number quickly increases with an increasing number of points. Thus, optimizing the route by means of brute (i.e. trying out every single solution) force quickly becomes very computationally heavy. In order to avoid optimizing by brute force, global optimization functions can be used in order to find the optimal solution. We have chosen to use the genetic algorithm, as it is easy to implement when doing discrete optimization, which this problem is. Discrete optimization means that there is a finite set of possible solutions, instead of a continuous range of solutions.

5.2 Genetic Algorithm

The genetic algorithm is a meta-heuristic algorithm inspired by the idea of ”survival of the fittest”. The basic idea is to create a ”generation”of sample solutions, by encoding the data of solutions as ”chromosomes”. New generations of sample solutions are then created by combining these candidate solutions, with the more ”fit”samples being more likely to be chosen for ”reproduction”. The idea is that attributes that result in good fitness will remain, while attributes resulting in bad fitness will be sorted out. Furthermore, each sample has a chance to randomly mutate, which reflects the random mutations happening in nature as well.

For each generation the best solution is found and is compared to the *global* best solution, i.e. the best solution encountered so far over all generations. If the best solution for the generation is better than the global best solution, the global best solution is replaced.

5.2.1 Objective function

In order to evaluate fitness of a sample, an objective function must be implemented. In our case, the objective function is the total route distance, given a certain route of points.

5.2.2 Fitness function

For each sample, the *fitness*, a measure of how good the solution is, is calculated. This measure is used when picking out the candidate solutions used to create a new generation.

In our implementation the fitness of each sample solution will be evaluated as the relative increase or decrease of the objective function for the given sample, compared to the current optimal solution. This is done in order to make the fitness independent of the scaling used to measure distance.

$$fitness(points) = \frac{dist_{total}(\mathbf{points})}{bestFitness_{k-1}} \tag{5.4}$$

where $bestFitness_{k-1}$ is the best fitness score experienced up until last iteration. In the first generation however, the fitness denominator in the fitness-function will be the max of the first generation.

5.2.3 Encoding of information

In order to create combinations of candidate solution, information must be encoded as a *chromosome*. An example could be a candidate solution consisting of a sequence of actions, where each action can be one of two possible actions. This could be encoded as a sequence of binary values.

In our example, the information of each candidate solution is the sequence in which the points are visited. We choose to encode this information as a sequence of letters. For example, one candidate solution is to visit point A, then point E, then point B, then point C. This would be encoded as A E B C.

5.2.4 Creation of new generation

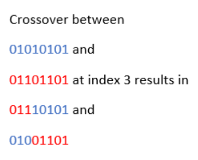
The creation of a new generation of sample solutions consists of several steps. Two sample solutions are from the current generation is selected for *mating* and then a sample in the new generation is created using crossover and *mutation*.

5.2.4.1 Selection of samples for mating

When creating a new sample solution, two sample solutions in the current generation is picked. When picking sample solutions for mating, the chance of a sample being picked depends on its fitness, with more fit solutions having a larger chance of being picked. Furthermore, it is important that the two picked solutions are not the same, as this would not result in a new sample solution, when doing crossover.

5.2.4.2 Crossover

When two solutions have been picked, two new solutions are created by means of a method called *crossover*. The idea of crossover is to take the chromosome-string from both solutions, "cut"them over at a given point, and then combine the resulting parts to two new solutions. For example, if the data for a solution was encoded as eight binary numbers, crossover could look as follows:


Crossover between

01010101 and

01101101 at index 3 results in

01110101 and

01001101

Figur 5.1: Example of crossover with information encoded as binary numbers

In our implementation, crossover is a bit different, seeing that each point can only appear once in a solution. This has been solved, by the following implementation of crossover between solutions A and B at index N:

1. The first N elements are taken from solution A.
2. The first element not filled out is equal to the first element in B not present in the current sample.
3. Repeat step 2 until all elements are filled out.

where "first" means lowest index in vector. The algorithm is repeated, where solution A and solution B swap places, resulting in 2 new solutions. An example where, information is encoded as letters is:

Crossover between

A B C D E F G and

B D C F A E G Results in

A B C D F E G

B D C A E F G

Figur 5.2: Example of crossover with information encoded as letters

5.2.4.3 Mutation

After generating new sample solutions from crossover, there is a chance that each sample solution will mutate, e.g. create a random change to the solution . This chance is decided by a variable mutation rate. In the case of the binary representation of information, mutation could be implemented as a chance for each bit to flip.

In our case, mutation is implemented as a chance to flip the position of two elements of the vector in the sample solution.

For example:

A B C D E F G could mutate into

A F C D E B G

Figur 5.3: Example of mutation with information encoded as letters

5.2.5 Stopping criterion

The algorithm must contain a stopping criterion, which is the criterion that must be fulfilled before the algorithm stops searching for a better solution. Possible stopping criterions are:

- Certain number of generations run.
- Certain number of generations without improvement in global best solution.

- Global best solution evaluates to more/less than certain value.

In our case, the stopping criterion will be a certain number of generations without improvement in global best solution.

Kapitel 6

Litteratur

[1] Wikipedia contributors. Travelling salesman problem — Wikipedia, the free encyclopedia, 2019. [Online; accessed 18-December-2019].