Optimized Shopping, solving TSP using a parallel Genetic Algorithm

Thomas, Sven
Gummersbach, Germany
sven.thomas@th-koeln.de

Schmeier, Philipp Felix
Gummersbach, Germany
philipp.schmeier@th-koeln.de

Kolodziej, Sheila
Solingen, Germany
sheila.kolodziej@th-koeln.de

Gezer, Ayhan
Cologne, Germany
ayhan.gezer@th-koeln.de

Yüca, Volkan
Cologne, Germany
volkan.yueca@th-koeln.de

Öztürk, Sümeyye Nur
Cologne, Germany
suemeyye_nur.oetuerk@th-koeln.de

ABSTRACT

This work deals with the problem of finding the shortest route during shopping using a shopping list. It is therefore important to find out which route is the best aka shortest route based on the products and their sorting in the store. A problem commonly known as a traveling salesman problem. In this paper, a solution is presented that is based on parallel genetic algorithms. To use the full advantages of the parallel genetic algorithms, a three-part architecture was designed. The architecture consists of a display component, which was implemented as a Unity frontend, and a master/worker architecture. The master component ensures communication with the frontend and manages the master population, which are processed by the workers.

Author Keywords

traveling salesman; tsp; genetic algorithm; parallel genetic algorithm; distributed systems; th köln; architecture of distributed systems

INTRODUCTION

This project is part of the Architecture of Distributed Systems module in the Master's program in Computer Science (specializing in software engineering) at the TH Köln. The aim of the module is to work on a problem that can be solved faster or better by using a distributed architecture. Students are free to choose the topic.

For implementation and testing, the students are provided with a PC pool with 12 Macs (insert TODO exact description). With the help of these computers, the Measuerment for this project were carried out.

Technology
Arts Sciences
TH Köln

PROBLEM DESCRIPTION

The selected problem is a classic traveling salesman problem. The aim is to use a shopping list to find the optimal route through a chosen store to enable the customer to have a stress-free shopping experience.

The solution developed in the project will be tested using the example of the Rewe market in the Gummersbach forum. The range and scale are abstracted to ensure a clear presentation.

ALGORITHM

When finding the algorithm, the main focus was on the possible parallelization, which made the choice of a genetic algorithm. The advantage of genetic algorithms is that they can be parallelized particularly well in a variety of ways. When calculating genetic algorithms, the most time-consuming operation is determining the fitness of an individual [1]. Since the distribution of the fitness calculation over a wide number of components would generate a large network traffic, the distributed calculation of entire populations was decided.

To ensure an exchange between many populations, different models, such as the island modell [2], the pollen model [3] and the nearest neighbor migration model [3, 4] were compared and analyzed.

The nearest neighbor migration was chosen for the algorithm because it offers the advantage of generating a low network load (especially compared to the pollen model) and at the same time ensuring a continuous exchange of individuals among the populations.

Further information on configuration can be found in the Configuration of the Algorithm chapter.

ARCHITECTURE

The architecture was divided into three components. Firstly, the pure frontend, which was implemented in Unity. Secondly, the implementation of the algorithm in its own master/worker architecture. The master, hereinafter referred to as the scheduler, ensures the exchange between the workers and with the frontend. The workers receive populations and evolve them.

In the design there was a strong focus on the loose coupling of the components so that the failure of individual components does not limit the functionality of the entire system.

REST was used for the exchange between the components. This guarantees a free choice of technology for the individual components (TODO source).

COMPONENTS

Unity-Frontend

The frontend was developed in Unity and, in addition to the visual display, is also responsible for generating the shopping list and a navigation mesh that shows all producty and usable walkways. After creating a shopping list, it is sent to the scheduler. By regulary asking the scheduler, new calculation results are queried and displayed.

Scheduler

The scheduler represents the master component and therefore offers a REST API, which can be viewed in the OpenApi 3.0 documentation. The scheduler is written in Kotlin and Ktor is used as the server framework.

After receiving a new shopping list, a master population is created. All workers who are already registered or who will register in the future will be assigned a subpopulation for processing. If a worker delivers an updated population, an individual exchange is initiated with the neighboring population.

Worker

The worker maps the computing component and is implemented as a simple Java application. At the beginning, the worker logs on to the scheduler and waits for a shopping list and thus population to be available. After receiving the population and the navigation mesh, the population is processed.

CONFIGURATION OF THE ALGORITHM

With genetic algorithms, there are many possible adjustments that can be tuned to influence the result. Different configurations were therefore tested for scheduler and workers.

The most important are named below and briefly explained. More detailed explanations and specific values can be found in the complete documentation (available only in german).

Scheduler-Config

- Worker-Count: The number of simultaneous workers supported by the scheduler. Based on this, the number of subpopulations is calculated
- Master-Population-Size: The size of the master population.
- Demo-Individual-Size: Before a final result is passed on to the frontend, a number of individuals is held, which consists of the best individuals from the processed populations. Only when this minimum number is reached is a result passed on.
- Min-Delta: The min delta ensures that no final result is passed on to the frontend as long as there are still large discrepancies between the demo individuals collected. So there is still a lot of room for improvement.
- Individual-Exchange-Count: This determines how many individuals are exchanged between the subpopulations.

Worker-Config

- Mutation-Rate: The mutation rate describes the frequency of the mutation of a single gene.
- Tournament-Size: The tournament size describes how many individuals are randomly drawn from the population for the selection.

MEASUERMENT

TODO Short description how did we measure. Fokus on results and configuration of scheduler and worker

CONCLUSION

TODO sum up: what have we done ;)

The complete code is available on GitHub.

ACKNOWLEDGMENTS

We thank the lecturers of the module, Prof Dr. Lutz Köhler and M. Sc. Pascal Schönthier for the support during the semester and the opportunity to write and publish this paper.

REFERENCES

[1] Fuey Sian Chong. 1999. Java based Distributed Genetic Programming on the Internet. Technical Report CSRP-99-7, School of Computer Science, The University of Birmingham. (1999). Retrieved November 17, 2019.

[2] Minc Duc Nguyen. 2017. *Optimierung von Routingproblemen mit Genetischen Algorithmen auf Apache Spark.* Ph.D. Dissertation. Hochschule für Angewandte Wissenschaften Hamburg, Hamburg, Germany. Retrieved January 02, 2020 from `http://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4164/pdf/MinhDucNguyen_Optimierung_von_Routingproblemen_mit_Genetischen_Algorithmen_auf_Apache_Spark.pdf`.

[3] Marion Riedel. 2002. *Parallele Genetische Algorithmen mit Anwendungen.* Ph.D. Dissertation. TU Chemitz, Chemitz, Germany. Retrieved January 05, 2020.

[4] Shisanu Tongchim and Prabhas Chongstitvatana. 2001. Parallel genetic programming synchronous and asyncronous migration. *Artif Life Robotics* (2001).