EuroCC@Turkey

https://eurocc.truba.gov.tr/

This document is prepared by EuroCC@Turkey for EuroCC under GA No 951732

CASE STUDY REPORT

Public transport timetable optimization on HPC

NCC Partner	*METU (Middle East Technical University)*
Company*	*Parabol Yazılım Ltd. Şti. – paraboly.com* *Abdulkadir Kılavuz – abdulkadir.kilavuz@paraboly.com* *Mehmet Barutçu – mehmet.barutcu@paraboly.com*
Expert	*Cevat Şener – csener@metu.edu.tr*
Start & End Date	*July 2022 – November 2022*

* **Parabol Yazılım Ltd. Şti.** accepts that the Case Study Report is shared with the EuroCC Project and the community through the EuroCC@Turkey awareness creation activities and platforms.

Contents

1. Problem Identification

Parabol Yazılım (paraboly.com) has been carrying out R&D activities in the intelligent transportation systems sector since 2011. As one of the main outputs of a TUBITAK 1501 project, METIS (paraboly.com/metis), an intelligent transportation system (ITS) management software, has been used in more than 25 cities with a 70% market share in Turkey. Parabol has also been developing Cermoni (cermoni.app) as a public transportation (PT) decision support system, to help improving public transportation system in cities by making informed decisions based on mobility analysis.

Optimizing public transport and exploring passenger mobility requires handling large amounts of data, solving complex combinatorial optimization problems, and dealing with uncertainty. Effective and scalable processing of high volumes of multidimensional data produced in public transport systems is one of the biggest problems. The proposed experiment will contribute to the robustness of a PT system via HPC infrastructure and ensure correct and fast decision-making by the authorities. It represents an important step for smart mobility and sustainable urban mobility plans.

This experiment aims to improve the timetable optimization algorithm on the TRUBA HPC platform. This algorithm tries to find the solution that best meets the OD requirement of the city with the number of vehicles and drivers available. The problem space is so wide that the execution time of the final version and the algorithm testing throughout experimental development will take a long. Apart from the problem space, another major challenge is about the target functions to optimize. Since authorities can make arrangements according to different criteria, a single function will not suffice to show the needs of these authorities. Although what is desired is to transport the maximum number of passengers at the right time and with minimum delay, it won't be easy to digitize and return it to a single function. Hence, an HPC infrastructure is needed to alleviate and solve these difficulties. Developing, running and debugging such a complex optimization algorithm on an ordinary server is not feasible, as it will take hours to get just a single solution.

2. First Suggestion

Our current version of the timetable optimization algorithm runs sequentially, considering one route at a time. Algorithm schedules all routes in an order and uses remaining resources at each step, such as vehicles, drivers etc. This greedy approach yields good results but still does not guarantee the optimal or even near-optimal solution since it does not consider all routes together in a terminal. Since the problem is known to be NP-hard and the computers we work with have limited resources, an optimization algorithm that considers all routes together does not yield an optimal solution in a feasible time.

We plan to port and run this optimization algorithm on TRUBA and make it parallel using DASK. We will follow the data-parallelism approach based on regional terminals and lines. Next, we will work on improving its success in terms of result optimality while keeping the execution-time reasonable. Methods to apply may include using costlier heuristics that can produce better timetables, employing a combinatorial approach instead of pure greedy, benefitting from parallel backtracking etc. We will, of course, compare our test results with optimal solutions to measure the performance of our approach.

The plan we will follow has the steps below:

Timeline	**Tasks and Milestones planned**
July	Orientation on the TRUBA platform, DASK and other related tools Set up and test the environment
End of July	Milestone: TRUBA platform experienced & Runtime environment tested
Aug	Port the current timetable optimization algorithm to TRUBA Make it data-parallel over regional terminals and lines on DASK
End of Aug	Milestone: First data-parallel version of the algorithm running on TRUBA
Sep - Oct	Improve the algorithm to achieve better-optimized results within a reasonable amount of time
End of Oct	Milestone: Improved version running on TRUBA

3. Solution Stage-I

Timetable optimization algorithm consists of integer decision values which cause long waiting times to solve the problem as solution space gets larger. We initially had our iterative algorithm to deal with this problem in our production environment. Since we have high-performance computers in the TRUBA environment, we had better resources for this optimization problem to work. In this case, to test the limitations of both our problem space and TRUBA environment, we first extended the algorithm for which it aims to achieve a global optimum solution rather than the iterative heuristic. Changes in the formulation are as follows:

- The "number of routes" parameter is added formulation to solve multiple routes of timetable problem at once.
- Multiple vehicle type parameter is added to support different types of vehicles having different capacities for assigning the suitable vehicle types for different time slices.

- The "optimality gap" option is added as a stopping condition when the model has difficulty finding an optimal solution.
- The "timeout" option is added to make it stop after a too-long execution.

After redefining the algorithm, we implemented the DASK platform for the parallelization of multiple depots to solve timetables related to each terminal at once. We chose DASK to parallelize algorithms because it is very time efficient to implement and not necessary to change the code of the original sequential algorithm. An example of the DASK implementation is given in Figure 1. DASK runs the algorithm in data parallel after defining the parameters and the tasks.

```
from dask import delayed, compute
lazy_results = []
for _ in range(5):
    lazy_result = delayed(RouteTruba(routes=R,time_intervals=time_intervals,terminal=K,departures=F,maxLoads=P,\
    desiredLoads=d,routeTimes=T,crewSize=250,lowerDeparture=L,\
        upperDeparture=U,M=999))
    lazy_results.append(lazy_result.solve())
results = compute(*lazy_results)
```

Figure 1: Sample Code from the DASK Implementation

We started with a toy example to test out our DASK implementation by defining relatively impractical parameters to reduce the computation time of the algorithms to make sure that our algorithms work in parallel. We containerized our code using Docker and sent them out to TRUBA platform using SCP.

After sending the necessary files, running on the platform was easy because of two reasons:

- There is almost no queue waiting times in the DASK queue.
- The implementation of singularity made it easy for running the tasks.

We run our toy examples on both our local computers and TRUBA platform and get the same results on both with comparable execution durations because of the small problem size

4. Solution Stage-II

We continued our study by running our algorithms with real data. Our initial assumption is that problem sizes could be so large that we may not reach an optimal solution in a feasible duration of time. For that, we added the "optimality gap" and "timeout" options to the algorithm as a terminating condition. We have three parameter sets defining their categories: small, medium and large. We could get an optimal solution for a small parameter set in our local machine in a few minutes, but for each of medium and large size problems, unfortunately, we could not find an optimal solution, even after waiting for 7 hours. We then tested out those parameter sets on the TRUBA platform.

As expected, we achieved the solution for a small parameter size in just a few minutes. We again got the solution for the medium size, but this time, it took around 2 hours for the algorithm to achieve that. We could not get optimality for the large size, but we reached the "optimality gap" stopping condition, which is set as 0.1. An example of an optimal solution for medium size problem is given in Figure 2.

```
Result - Optimal solution found

Objective value:                22072.50000000
Enumerated nodes:               45850
Total iterations:               1250268
```

Figure 2: Result of a Medium Size Problem

5. Results and Achievements

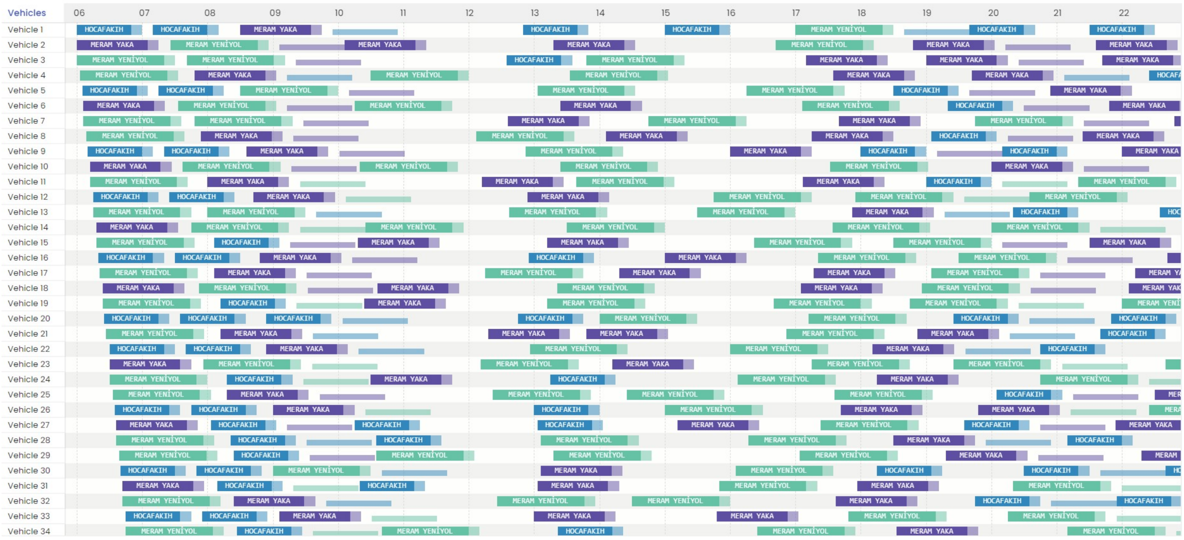
In this case study, we have successfully used the TRUBA HPC environment for the second time. As a continuation study for public transportation O-D analysis, processing that information through optimization added much value to our case. We, as a company, also learned a lot about HPC environments, DASK parallelization and cloud operations during our experiments. We also had a chance to share our experiences with our colleagues at the event "Industry Relation Day".

We extended our heuristic algorithm to optimal during our experiments and tested it on multiple scenarios. First, we tested small-size problems on both the local computer and HPC platform to ensure that parallelization was working well. Then, we extended our problem size and found an optimal solution that could not be possible on a local machine. Last, to see our algorithm limitation and HPC performance, we ran a large-size problem and tried to find an optimal solution. Unfortunately, since the problem is NP-Hard, we could not reach optimal but found near optimal with a gap as small as 0.1. Even though the optimality has yet to be reached for this size, we still gained much information on a near-optimal solution.

For the implementation side, we have not faced any difficulty in implementing DASK on the TRUBA platform. Singularity integration at the platform made our implementation more user-friendly and easier to understand. In addition, DASK queues were shorter, resulting in shorter waiting times for execution and getting solutions.

The result of these analyses can be used in optimizing the resources of municipalities when reducing the operation cost and increasing the service level in cities. A sample timetable of Konya optimized using the algorithm developed is shown in Figure 3. It is more vital than ever, especially considering the inflation in fuel costs and the operational cost of vehicles. Our study will contribute to an increase in the efficiency and quality of public transportation.

The TRUBA HPC environment made a significant contribution to our R&D cycle. Thanks to it, we could test the changes in the algorithm much faster. TRUBA enabled us to do experiments that could not be possible on our production servers.



HOCAFAKIH TIP FAKÜLTESİ ANASULTAN	MERAM YAKA ANASULTAN	MERAM YENİYOL DÖRTOKKA
06:00	06:00	06:00
06:05	06:06	06:03
06:10	06:12	06:06
06:15	06:18	06:09
06:20	06:24	06:12
06:25	06:30	06:15
06:30	06:36	06:18
06:35	06:42	06:21
06:40	06:48	06:24
06:45	06:54	06:27
06:50	07:00	06:30
06:55	07:06	06:33
07:00	07:12	06:36
07:05	07:18	06:39
07:10	07:24	06:42
07:15	07:30	06:45
07:20	07:36	06:48
07:25	07:42	06:51
07:30	07:48	06:54
07:35	07:54	06:57

Figure 3: A Timetable Optimized