Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Making the next step in finding the best route

Ajda Zavrtanik Drglin[a,*], Romi Koželj[a], Martin Pečar[a], Gregor Mrak[a]

[a]OmniOpti d.o.o., Celovška street 189, 1000 Ljubljana, Slovenia

Abstract

Because the vehicle routing problem (VRP) is a complex topic, with practical cases being too large to obtain the optimal solutions, heuristic approaches need to be used. Here, High-Performance Computing (HPC) provides computing and memory resources that can accommodate the size of these cases. In this paper, we describe two methods, namely the multi-seed computation and the step method and test their performance on a realistic VRP example from a logistic company. Since logistic companies need to have a good result as fast as possible, we were mainly interested in finding the best possible solution in a given time frame or after a certain number of iterations.

1. Introduction

This paper describes the project of using "HPC Logistics" along with its results. The main focus of the project was to determine whether a high performance computer could significantly improve the process of finding a good solution with the use of the optimization engine, developed for solving vehicle routing problems of different difficulties, either by reaching a better solution in a fixed time frame or by reducing the time needed to find a good solution, and discover ways for which the benefits are highest.

Even though the optimization engine used in this project employs state-of-the-art search strategies, obtaining good results is still very time consuming and there is no guarantee that we will indeed receive a good solution. This is due to the fact that VRP is an NP-hard problem, which means that it can only be optimally solved in a reasonable amount of time for very small problems; for any real-life situations, the optimal solution is not known and heuristic approaches need to be used in order to obtain a "good" solution. Here, the notion of a "good" solution is somewhat subjective, since the optimal solution is usually not known and therefore we do not have a suitable benchmark. In the discussion of the results of this paper, we will say that a solution is "good" if its cost (the result of the objective function, discussed in detail in the next section) is at most 5 % or 1 % higher than the cost of the best solution ever found, depending on the context. The effort needed to obtain such a solution depends on the size of the problem and number of constraints of the VRP, for instance, the more vehicle capacity dimensions there are, the slower the optimization algorithm is. The cost of the solution decreases fast in the beginning, but after that more and more time is needed to obtain an improvement, if it exists. This means that for larger logistics companies with several hundred vehicles in their fleet, getting a good solution could take many hours, which might be too much to be useful. Our objective therefore is to take advantage of all the resources currently at the disposal to increase the chances of obtaining a good solution in a fixed time frame.

The process of optimization starts with an initial solution, which can be given or generated with a heuristic method. The optimization process is then steered by a seed, which can be given as an input, and it serves as the seed for the internal pseudo-random number generator. Different seeds determine the process of optimization (as well as the creation of the initial solution) – they are used to make noise, for example to temporarily accept a worse solution in order to escape a local minimum, and some job set creations to improve the behaviour of the algorithm – and in turn the solution cost has different convergence speed. All seeds are not guaranteed to lead to a good, let alone optimal, solution in any amount of time. A certain seed may lead us closer to the optimal result than others or it might achieve this in a shorter amount of time. In addition, some seeds may perform better in the beginning of the

*Corresponding author, e-mail: ajda.zavrtanik@omniopti.si

process, while others perform better later on. We say that a seed is "good" if the computation with this seed yields a good solution before the algorithm reaches its termination criterion.

1.1. Employing HPC

As we currently cannot predict which seeds are better for certain problems or for certain stages in the optimization process, we took advantage of the resources of a high performance computer and ran many computations at once, each with a different seed. This way we increased the chances of finding a good seed for a certain problem while mimicking the conditions of a logistics company, which does not have access to machines with a large computing capacity, but has many limited capacity computers. The computer used in this project consisted of 14 nodes, each of these nodes had 2 Intel Xeon E5-2690v4 processors, together having 28 cores clocked at 2.60 GHz. Every node also had 512 GB of fast DDR4 RAM. We ran optimization processes simultaneously, one per core, which ensured us that the computing time for the individual processes did not increase – this would happen if the hardware resources had to be shared between the processes. Based on the resources available, we applied two simple techniques, explained in the next section.

2. Tools and methods

The optimization engine used in this project is based on the open source project jsprit [1] with some additional features added (for example additional problem constraints, more flexible time and distance cost). jsprit is a java based, open source toolkit, equipped for solving such rich travelling salesman problems and VRPs [2, 3]. It is lightweight, flexible and easy-to-use, and based on a single all-purpose meta-heuristic, which is best suited for complex problems with many constraints and is simple in structure. The meta-heuristic creates new solutions, which are evaluated with an objective function that tells us the solution's *cost*. jsprit lets the user define cost parameters for multiple aspects of the solution (such as cost per distance and time unit, cost per stop, cost per unassigned job, cost per vehicle used) and the solution cost is the sum of all the specified costs (for example, if one sets all the cost parameters to 0 except for cost per kilometer, which is set to 1, the cost of a solution with 100 km driven in total is 100). The cost is considered to be without units as we are only interested in its value, but it can be interpreted as financial cost in a monetary currency, if the cost parameters correspond to actual costs for each parameter (this is not always the case – the user can use different cost parameters to achieve different outcomes, for example a higher cost per time unit will result in a time-wise quicker solution, whereas a higher cost per distance unit will result in a solution with less distance driven, and these two solutions might differ). The cost enables us to compare solutions to one another directly, and we say that one solution is better than the other if its cost is lower. In this paper we used cost parameters that have proven to work well for most of the VRPs.

The meta-heuristic used by jsprit was developed by Schrimpf et al. [4] who formulated the *ruin-and-recreate* principle – a large neighbourhood search that combines elements of simulated annealing and threshold-accepting algorithms. The algorithm either accepts or creates an initial solution and after that works in iterations to try to improve it. Each iteration consists of two parts: ruin and recreation. First, parts of the solution are disintegrated, leading to (I.) a set of jobs that are not served by a vehicle anymore and to (II.) a partial solution containing all other jobs. Thus, this is called the *ruin* principle. Based on the partial solution (II.) all jobs from (I.) are re-integrated again, which is therefore referred to as *recreation* [5], yielding a new solution and ending the current iteration. If the new solution is feasible (does not break any constraints) and has a suitable cost, it is accepted as the new best solution, whereupon a new *ruin-and-recreate* iteration starts. These steps are repeated over and over again until a certain termination criterion is met (e.g. computation time, number of iterations, etc.). One such iteration takes a relatively constant amount of time to compute for a certain problem (on a particular machine with its specifications), but this time differs greatly between problems of different sizes and difficulty. The number of iterations needed to obtain a satisfying solution also varies – for small problems around 100 iterations is usually enough, for bigger problems we need at least 1000 iterations.

To take advantage of the numerous cores at our disposal when using a high performance computer (while at the same time still mimicking a realistic situation of a logistics company where the process could be spread over independent machines), we developed two techniques to try to bypass the fact that we cannot determine whether a seed used in a computation is good or not.

- **Multi-seed method**: We have simply run several threads, each one with its own, different seed. This seed impacts the rate with which a better solution is found. It is impossible to anticipate which seed will give us better results for which problem. By using several processes with different seeds we increase the chances of finding a good solution within a given time, albeit using more computing power.

- **Step method**: We have tried to take advantage of the fact that some seeds were more "efficient" than the others, i.e. their convergence was faster. Our idea was to use the currently best known solution as an initial solution for the rest of the process. The process consists of multiple steps, each comprised of the following:
    1. Run the algorithm with multiple seeds as above.
    2. Stop the algorithm after a certain number of iterations and find the best solution found so far among all the seeds.
    3. Select this solution, set it as the initial solution (starting point) for all the seeds, and continue to 1.
    4. Repeat until the termination criterion (number of iterations, time elapsed) is met.

Multi-seed computation is a special case of the step method where the number of iterations for each step equals the number of iterations after which the algorithm should terminate, so we only have one step.

The idea for our step method arose from existing approaches:

- Gradient descent – where one uses the best solution as the start for the next step. The difference is that we are not able to analytically determine the best "direction" for continuation, but have to heuristically search the neighbourhood, using several threads with different seeds.
- Genetic algorithms – where a single step produces a "population" and the best representatives of the population are used for the next step. Here, the difference is that we have used only one best solution for the next step. We could potentially also use all non-dominated solutions, in which case we would need a new objective function to obtain another measure for the solution. This leads to multiple questions: what should this new objective function look like? Which seeds get which initial solution in the next step? The obvious candidate for the new objective function would be the usual cost function with different cost parameters. The two straightforward answers to the second question would be to either assign the initial solutions randomly or to continue the computation with every seed receiving every initial solution (this would lead to increasing the number of processes exponentially with each step). We leave these generalizations to future research.

3. Results

For the test cases we used a realistic VRP example from a logistics company. Test cases consisted of a problem with cargo and vehicle capacities with three dimensions, time windows and shipments with both pickup and delivery locations specified. We chose a problem for which we have previously (with standard methods) found a solution that was a plan with no unassigned shipments (even though sometimes the initial solutions could not deliver all shipments), but that was difficult enough that the algorithm could run for many thousand iterations or many hundred minutes and we might still obtain better solutions. The problem contained 556 shipments and 15 available vehicles. The best solution found for this problem had the cost of 65 220; for analysis we take a look at when solution costs go below 70 000 as well as when they fall below 105 % and/or 101 % of the best solution cost. Since this is a NP-hard problem, we do not know the cost of the actual optimal solution and therefore can only do such analysis after all the computations have terminated – the concept of a good solution can only be defined after that. In the case presented in this paper, the best solution was found with the step method and we use its cost for analysis of both the multi-seed and step method.

3.1. Initial testing and Multi-seed computation

We first ran the algorithm on 6 cores of the HPC system using 6 different seeds and calculated an average cost across all 6 seeds. We let the algorithm run for 200 minutes, mimicking a real-life situation where a logistics company needs to obtain results in a fixed amount of time. Then we repeated these calculations on 24 cores using 24 seeds for comparison. Results can be found in Table 1. We see from Table 1 that the best solution cost is lower for calculations using 24 seeds and the average best solution cost is comparable between the two cases, whereas the overall best solution cost is lower for calculations with 24 seeds. In the 24 seed calculation 10 out of the 24 seeds produced worse than average solutions and 14 seeds produced better than average solutions. The variations in average times show that not all seeds are good, but choosing multiple of them will increase the chances for one of them to be good (as well as bad) for a given problem. We can also see that not all seeds were successful enough to get a solution within 1 % or 5 % of the best solution (all 6 seeds in the first computation were good, whereas in the second computation only 22 of the 24 were good; only one seed out of all 30 was able to obtain a solution with the cost within 1 % of the best ever), which also shows that some seeds were not suitable for solving the problem – increasing the number of seeds also increased the chances that some of them will be unsuitable for this particular problem. On the other hand, best times are all significantly lower for calculations using a higher number of seeds.

This shows that calculations running with a higher number of seeds can obtain better solutions significantly faster, but it is crucial to try out multiple different seeds – the more seeds we try, the better our chances of finding a good solution fast. We should also point out that the net computation time equals 200 minutes times the number of seeds; however, this does not concern us, since our objective is to find a good solution using all resources available to us.

To check that claim more thoroughly we also ran calculations using 1, 3 and 12 seeds and plotted the best solution cost progression through time for each of these calculations (including the ones run with 6 and 24 seeds). Seeds were chosen randomly for each calculation separately. Results are shown in Figure 1. We can see from Figure 1 that calculations run with a higher number of seeds overall converged faster and to solutions with lower costs than calculations that were run with lower number of seeds. We can again see that a choice of seeds affects the calculation process, since the final best solution cost for 12 seeds is lower than the one for 24 seeds – indeed it would even be possible to be "lucky" and get the best possible solution in the 1 seed computation, but in general, the more simultaneous computations with different seeds you run, the higher the chance that one of them will be good. We could find no obvious pattern as to which seeds yield good solutions.

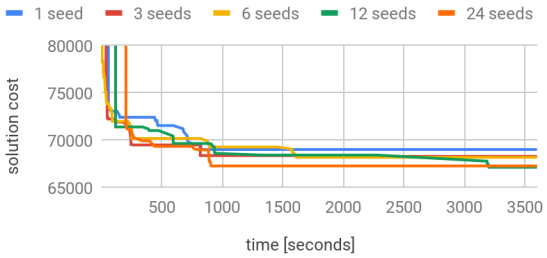

Figure 1. Best solution cost progressions for different numbers of seeds.

3.2. Step method

For the step method, instead of limiting the computation time like in the multi-seed method, we limited each step by the number of iterations to ensure fairness between cores since in this method cores exchange information (in the analysis we still consider the best solution cost progression through time since that is the crucial factor in the usual real-life scenarios). We ran the tests on the HPC system with 6 and 24 different seeds (in cases where we ran multiple computations with the same number of seeds, the seeds remained the same). For the computation with 6 different seeds, we set the total number of iterations to 1 000 and each step to 100 iterations (a total of 10 steps). For the 24-seed computation, in addition to the same configuration as for 6 seeds, we also tried a calculation with shorter steps, where we set the total number of iterations to 1 000 and each step to 10 iterations (100 steps) and one with longer steps where we set the total number of iterations to 10 000 and each step to 1 000 iterations (10 steps). With this approach we tried to determine an optimal step size. Results and comparisons can be found in Figure 2 and in Table 2.

Figure 2 shows the best solution cost progression through time for the computations with 24 seeds, where each of the computations used step sizes of different sizes. We show only 1 000 iterations for each of the computations. The important difference from the multi-seed method is that we enabled different cores to benefit from a better starting point, the initial solution, that might have been found by another core. We can see from the upper left figure of Figure 2 that this shared knowledge was not used in this computation, where size of the step is 1 000

Table 1. Results of the multi-seed computation.

Nr of seeds	Average best solution cost	Best solution cost	Average time needed for best solution [s]	Best time to break cost of 70 000 [s]	Average time to break cost of 70000 [s]	Best time to get within 5 % of best solution ever [s]	Average time to get within 5 % of best solution ever [s]	Nr of seeds that got a solution within 5 % of best ever
6	66 947	66 073	5 631	190	524	284	1 268	6
24	67 009	65 790	5 628	116	1 018	208	2 002	22

iterations – separate cores did not receive any help from one another, which resulted in three of them with a best solution cost of almost 80 000 after 1 000 iterations. Results in the upper right figure and bottom figure of Figure 2 show that shorter steps significantly increase the convergence speed in the beginning of the process, but steps that are too short, such as in the bottom figure, disable the search of the solution space more in-depth, which in turn slows down the best solution cost convergence after the initial fast convergence, when a quick search is enough.

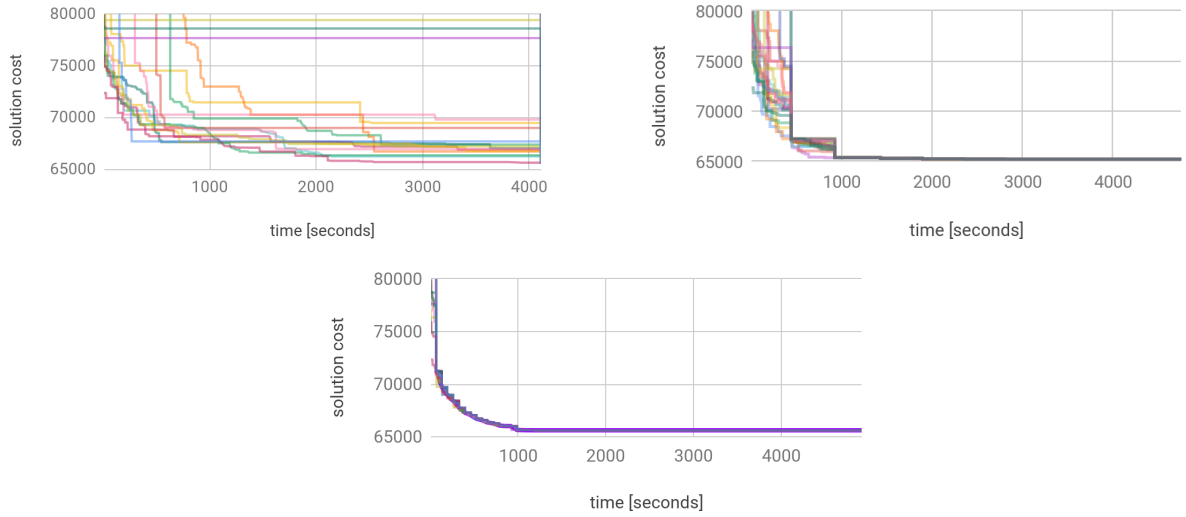

Figure 2. Best solution cost progressions for 1 000 iterations where one step consisted of all 1 000 iterations (upper left - 1 step), 100 iterations (upper right - 10 steps) and 10 iterations (bottom - 100 steps).

From Table 2 we can see that shorter steps are very effective in providing good results very fast, while longer steps may eventually bring us closer to the optimal solution. If we compare results in columns with the best solution cost after 1 min, 5 min, 15 min and 30 min we can see that the computation with 10 iterations per step achieved a significantly better result in the first minute, but this effect disappeared already after 5 minutes, when this computation had the worst result (its final result was also the worst). We can see that at the 5 minute mark the computation with 6 seeds had the best result, which we can attribute to a good choice of one of the seeds; this computation was also the fastest in reaching the 5 % of the best solution cost mark (for the same reason). After 15 minutes the computation with 24 seeds and 100-iteration steps had a considerably better solution than the other computations (even though the 6-seed computation had a head start due to a lucky choice of seed); this was also the process which yielded the best solution ever. We can also see that this computation got a result within 1 % of the best solution cost in considerably less time than any other computation. The computation with 1 000-iteration steps obviously ran the longest since it had 10 times the number of iterations than any other, and while it did produce a very good solution, it needed the most time out of any computation to reach the 1 % of the best solution cost mark. Knowing that the solution cost in the optimization process decreases fast in the beginning and then ever

Table 2. Results for the step method.

Total nr iterations	Nr iterations per step	Nr seeds	Total computation time [s]	Best solution cost	Computation time for best solution [s]	Best solution after 1 min	Best solution after 5 mins	Best solution after 15 mins	Best solution after 30 mins	Time to get within 5 % of best solution ever [s]	Time to get within 1 % of best solution ever
1 000	100	6	3 712	65 401	2 563	71 570	67 311	66 031	65 478	160	953
1 000	100	24	4 756	65 220	4 273	71 844	67 674	65 379	65 270	208	565
1 000	10	24	4 915	65 630	1 362	71 285	67 784	66 027	65 630	222	923
10 000	1 000	24	41 767	65 268	20 718	71 844	67 674	67 600	66 592	266	2 108

more slowly, an idea for further work is to start with smaller steps which are then increased in order to enable finding better solutions faster in the beginning and more efficiently later in the process.

If we compare the multi-seed method with the step method, we can quickly see that the step method is more efficient. The total computation time of all three step method calculations with a total of 1 000 iterations was less than 82 minutes and all of them produced better solutions than the multi-seed computations, which ran for 200 minutes. In fact, if we take a look at the best solution cost of these three computations after only 30 minutes, we can see that all the best solution costs are lower than the best solution cost of the multi-seed method, which means that in this case we could reduce the computation time by 85 % and still obtain a better solution. This illustrates the effect of sharing the knowledge of the current best solution between cores.

4. Conclusion

We were interested in obtaining good solutions for VRPs as fast as possible. We have tried some approaches, which produced good results.

Multi-seed method: We can conclude that with a bigger number of seeds (and corresponding threads) chances of finding better solutions are higher, since also the probability that one of the seeds will be good for a given problem is greater.

Step method: We have developed this method to improve the performance, to enable finding good results faster. Each step resets the initial solution to the best solution found so far. The convergence becomes much faster, especially in the beginning.

If we compare the multi-seed and step methods we can see that in the step method provides some clear advantages over the multi-seed method. Even though both methods performed comparably well in terms of time needed to get a solution cost within 5 % of the best solution cost, the multi-seed method yielded worse final results, with only one of the seeds succeeding in obtaining a solution within 1 % of the best solution cost – all the other cores were essentially wasting resources. With the step method, all computations succeeded in obtaining a solution within 1 % of the best solution cost, most of them within the first 16 minutes (for comparison, the multi-seed computations ran for 200 minutes).

Future work: We are interested in determining optimal step sizes, perhaps using smaller steps in the beginning and then increasing their size. The next idea is to try with several different initial solutions, not just a single one, at each step. And the hardest problem, if solvable at all, would be to determine suitable seeds for specific problems.

Glossary

heuristic is a method for speeding up the solving of a specific problem or finding a good approximation to its solution. 1

iteration is a part of the optimization algorithm, consisting of the ruin (disintegrating parts of the solution) and recreate (inserting the disintegrated parts back in the solution in a new way) parts. Optimization process usually consist of multiple hundreds iterations. 2

meta-heuristic is a general method for speeding up the solving of difficult problems or finding a good approximation to their solution. 2

multi-seed method is a method for speeding up the optimization process with multiple simultaneous computations. 2

step is one part of the step method, consisting of running the optimization for a certain number of iterations (size of the step) for each seed, determining the best solution and setting it as the initial solution for all the seeds in the next step. 3

step method is a method for speeding up the optimization process with multiple simultaneous computations and multiple steps, within each of them a best solution is found and set as the initial solution for the next step for all computations. 3

vehicle routing problem is an optimization problem which asks "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?" . 1, 7

VRP see vehicle routing problem. 1

References

[1] Stefan Schröder. (2014). jsprit. `https://jsprit.github.io/` [Accessed 12.07.2021]

[2] Dantzig, George Bernard., Ramser, John Hubert. (1959). The Truck Dispatching Problem (PDF). Management Science. 6 (1): 80–91. doi:10.1287/mnsc.6.1.80.

[3] Toth, P., Vigo, D., eds. (2002). The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. 9. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 0-89871-579-2.

[4] Schrimpf, Gerhard, et al. (2000). Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159.2. 139-171.

[5] Pisinger, David., Ropke, Stefan. (2005). A general heuristic for vehicle routing problems. Computers & Operations Research. 34. 2403-2435.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU's Horizon 2020 Research and Innovation programme (2014-2020) under grant agreement 823767. The work was achieved using the PRACE Research Infrastructure resources at the CURIE (GENCI), the FERMI (CINECA), and the SuperMUC (LRZ) supercomputers.