A Parallel Hybrid Genetic Search for the Capacitated VRP with Pickup and Delivery *

Timo Stadler[1], Spyro Nita[2], and Jan Dünnweber[1]

[1] OTH Regensburg, 93053 Regensburg, Germany
[2] EPCC, Edinburgh EH8 9BT, United Kingdom

Abstract. In the realm of parallel computing, optimization plays a pivotal role in achieving efficient and scalable solutions. In this work, we present the parallelization of a hybrid genetic search for solving the Capacitated Vehicle Routing Problem with Pickup and Delivery (CVRPPD). The hybrid algorithm combines a customized version of local search with a genetic algorithm to compute an effective solution. Our implementation makes use of the Message Passing Interface (MPI) for data distribution and parallel execution. In addition, we run multithreaded processes on NVIDIA graphical processors using the CUDA technology, which further increases the computation speed and consequently minimizes the runtime. Parallelization also allows the best improvement strategy to be used instead of the first-improvement strategy while maintaining the same runtime. We store the resulting routes in a bus route database which we created as the basis of an extensive library of optimal routes for our specific use case of optimizing bus routes in a rural area. The experimental results on real road data show that the parallel implementation of the Hybrid Genetic Search (HGS) achieves significant improvements in runtime over the sequential implementation above a certain problem size. We believe that our implementation of the parallel hybrid genetic search method can have a great influence on optimization strategies in parallel computing and can also be applied to other subproblems of the VRP.

1 INTRODUCTION

The traveling salesman problem can only be solved using heuristics for larger numbers of points as they are present in any real-world transportation system. As with most NP-hard problems, approximate solutions are computed which require a lot of computation time until the results are acceptable, especially when dealing with large problem instances. However, approximate methods are inevitable, especially for managing resources like drivers and vehicles, where an increase in efficiency quickly leads to considerably reduced costs.

* The work has been performed under the Project HPC-EUROPA3 (INFRAIA-2016-1-730897), with the support of the EC Research Innovation Action under the H2020 Programme; in particular, the author gratefully acknowledges the support of Adrian Jackson from the EPCC and the computer resources and technical support provided by EPCC

In our preceding research, we found out that a Hybrid Genetic Search (HGS) is suitable for solving a specific instance of VRP, the Capacitated Vehicle Routing Problem with Pickup and Delivery (CVRPPD) [1]. The CVRPPD adds two constraints to the VRP that need to be considered. The first constraint is that each vehicle used only has a certain passenger limit and cannot accommodate more. It should also be noted that each vehicle can have a different size. The second constraint extending the classic VRP is that the CVRPPD divides stops into pickup and delivery points for passengers. Passengers are not arbitrary goods delivered to interchangeable destinations from a common depot, but they have individual starting points and destinations. Therefore, the pickup and delivery constraint has multiple implications. On the one hand, the order in which a person is picked up and dropped off by a vehicle must be in the correct order. In addition, the delivery must be performed by the same vehicle as the pickups. In our paper "A Hybrid Genetic Algorithm for Solving the VRP with Pickup and Delivery in Rural Areas", we introduced an adapated gene transfer limiting the amount of possible mutations in each generation. This lead us to an algorithm for solving the CVRPPD for the special case of more than 200 bus stops, as typically found on the countryside.

Unlike common algorithms for solving the VRP or variations of it, we used a hybrid genetic algorithm. This means that we combine the optimization techniques of a genetic algorithm, such as crossover, mutation and selection with the common local search method for solving the problem. In Local Search, a number of metaheuristics in the form of so-called moves are used to determine the most optimal route. In our case, the length of the route is used as an optimization criterion. However, it is also possible to optimize for other parameters, such as the shortest possible travel time for all vehicles, by making simple adjustments. Even a Local Search can already be parallelized by using multiple initial solutions and each thread computing an optimization for its initial solution. By adding the extra layer of the genetic algorithm on top of the local search, the presented algorithm can achieve an even better optimum. In contrast to a parallelization of the local search, an exchange of the continuously optimizing solutions takes place here due to the crossover.

The presented algorithm differs from the current methods for solving the VRP in that it can find a solution to the CVRPPD through an evolutionary approach without permanently generating solutions that are infeasible. Adjustments were made to the execution of the Genetic Algorithm that allowed the constraints of the CVRPPD to always be met, thus allowing the problem to be solved by an HGS.

The use of a GA instead of heuristics allows us to parallelize the algorithm. Moreover, GA finds mutiple close-to optimal solutions simultaneously. If one of the solutions no longer matches the real travel time due to the current traffic volume, another one can be chosen. Furthermore, less information is required compared to using a heuristic and a lot less finetuning of parameters is required.

Despite of the fact, that countryside bus stops are less densely positioned than in the city, it is difficult to provide a reliable and frequent public transport

system between them at affordable costs. Individual settlements can be placed in the middle of wastelands or in villages with a very low population density, where it is difficult for public transport providers to operate economically. The less-than-ideal solution is that means of transport run on fixed routes, only irregularly or temporarily not at all. Some residential areas on the countryside cannot even be reached by public transport due to their location. For many residents, this means, that they need alternatives to public transports, which are typically more expensive and not environmentally sustainable, such as cabs or their own cars.

In an ongoing cooperation with the "Roding transportation services" and other industry partners, we are currently establishing an on-demand transportation service with so-called floating busses. With the help of dynamic route planning, these buses will be available wherever they are needed and transport people to and from the existing fixed routes. In this way, the public transport system can be expanded to meet people's demands. The dynamic routing can react quickly to changing conditions, such as increased demand, road works or other traffic obstacles, and minimize the travel time for each passenger [2]. This system should lead to a more often used public transport service for the countryside and an increase in customer satisfaction. Our algorithm is responsible for repeatedly calculating the required routes in the shortest possible time in order to take into account all the passengers requests and provide them with the data for their trips, such as travel time, departure time, and start and end stops, with minimal delay.

The algorithm is subsequently tested on a dataset containing 2000 stop points. While only a few of these locations have structural installations, such as a waiting bench, the 2000 positions are without exception locations around Roding where our bus service stops on demand. Roding is a small town in Bavaria, Germany with a bit more than 10000 inhabitants and an area of $113km^2$. We also considered the significantly less populated neighboring regions, covering an area of $674km^2$, for the placement of the stops.

Due to the population size, our case study assumes that in a utopian scenario all residents would want to use public transit at the same time and thus a maximum of 10,000 pairs must be calculated. However, the problem is not limited to this size on the part of Local Search and the Hybrid Approach and can be extended to an even larger number for other case studies in the future. Thus, all the ideas demonstrated in this paper can be applied to other case studies as well. Thus, our hybrid algorithm is not only useful for solving the rural VRP, but can be applied to any specific CVRPPD.

To prevent our route optimization software from falling into a local optimum, a GA was used. This is based on the theory of evolution, whereby a pool of solutions is optimized with each iteration and thus a set of optimized solutions is generated, which naturally requires more computational power than the creation of one single route suggestion. Therefore, we decided to parallelize the algorithm. We show that the best performance is possible, when only parts of the procedure run in parallel. It must be taken into account that data exchange

and coordination in parallel processes result in a corresponding overhead, which in the case of arranging only a few bus stops along a route, can lead to a slowing down of the program instead of an acceleration.

Using the Cirrus Cluster at EPCC in Edinburgh, we calculated all possible combinations of routes between the individual stops. Thus, we were able to store all possible tours and the combination of individual trips in a database table. Thus, the online booking system which we provide to the population of Roding and its surroundigs in cooperation with our industry partners, does not need access to an HPC platform for finding routes. The requested routes are instead selected from our database, which we pre-fill with optimized route suggestions. We are currently making our implementation of the route optimization algorithm available on our Github to make it available to other developers. Thus, ohter tour databases for other areas than Roding can be build using our methods and also other booking applications can retrieve pre-optimized routes quickly when needed. In order to take current changes in road construction into account, the algorithm must of course be re-executed at regular intervals.

By using an HGS, parallelizing it, and using CPU threads and GPU cores, our algorithm enables a faster solution to the CVRPPD. Most importantly, by using a *best-improvement strategy*, which takes as much time as the otherwise common *first-improvement strategy* due to parallelization, our algorithm achieves an optimum in less time than common approaches. Due to the Genetic Algorithm it is also prevented that the found solution is a local optimum. Especially for very large problems starting from a number of more than 400 points, which have to be approached, our algorithm achieves a shorter runtime than common sequential approaches. This is mainly due to the hardware used. For the experiments, the Cirrus network of the EPCC was used [3].

Section 2 describes further possibilities for the efficient solution of the CVRPPD, and subsequently discusses benefits through the parallelization of a hybrid genetic algorithm.

Section 3 describes the functionality of the HGS and deals in particular with the parallelizable sections of the program. Section 4 describes our implementation using MPI and CUDA. In section 5 the results of the parallelized algorithm are compared to those of the sequential algorithm for different problem sizes and evaluated. Finally, a summary of the results of this work is given.

2 RELATED WORK

There are different approaches to solve the CVRPPD. In general, exact and approximate methods are distinguished. Within the approximate methods, a further distinction can be made between the heuristics and the meta-heuristics. An example of a heuristic is the Clarke-Wright-Savings method, in which a sorting of the travel distances takes place according to length and pairs with the largest savings are combined to tours [4]. The Genetic Algorithm we use in conjunction with local search is one of the meta-heuristics. Here, several heuristic

methods are combined in an iterative process to find the most optimal solution to the problem [5].

In addition to other possibilities for code optimization, such as algorithmic improvements or memory caching, parallelization in particular can significantly reduce the runtime of a program. Simultaneous processing, by splitting the code across multiple threads or processes, can achieve a significant performance boost. However, careful implementation is also necessary to prevent conflicts between threads and to make the best use of the available resources [6]. The Cirrus Cluster at EPCC has 38 GPU compute nodes each equipped with 4 NVIDIA V100 (Volta) GPUs. CUDA provides a runtime environment that allows applications to run on the cores of a GPU [7]. For the parallelization of our algorithm, we decided to use MPI for communication among the compute nodes and additionally, the CUDA interface, for outsourcing computations to the GPUs.

Yelmewad and Talawar use a parallel version of the Local Search heuristic, for solving the Capacitated Vehicle Routing Problem (CVRP) [8]. They also used GPU-based parallel strategies to improve the runtime of the so-called individual moves performed within a Local Search. However, their work deals with CVRP, which has significantly fewer constraints and can lead to a solution that is infeasible for our public transport problem. Also, only the parallelization of the Local Search is done. By using the genetic algorithm, it can be prevented that a local optimum is reached [9]. The authors state that 99% of the algorithm's time is spent in improving the feasible solutions.

Our algorithm is called a Hybrid Algorithm because it is a combination of Local Search and a Genetic Algorithm, and combines the strengths and approaches of both techniques to improve efficiency and performance in solving the optimization problem. Through this combination, the hybrid algorithm benefits from the exploration capabilities of the genetic algorithm to find global optima and the exploitation capabilities of local search to refine these solutions in their vicinity. As a result, our hybrid algorithm can often find better solutions faster than the individual components alone. This makes it a powerful technique for solving complex optimization problems.

In "A Multi-GPU Parallel Genetic Algorithm For Large-Scale Vehicle Routing Problems" Abdelatti et al. consider solving VRPs using GAs on high-performance computing (HPC) platforms with up to 8 GPUs. The authors focus on VRPs with up to $20,000$ nodes. To achieve the maximum degree of parallelism, each array of the algorithm is mapped to block threads to achieve high throughput and low latency [10].

A large number of nodes must also be processed in our implementation of the CVRPPD. Moreover, for our public transport problem, not only a single vehicle has to be routed optimally, but multiple vehicles, each with a different passanger capacity. Taking into consideration that we deal with an online system, a high number of "live"-requests for routes to be planned must be expected, even if most of them are only requested and not booked eventually.

Another interesting way to improve the solution of the VRP is the use of neural networks [11]. Local search and crossover of an HGA can benefit from

machine learning algorithms. A graph neural network (GNN) representing a heat map is used as a more efficient alternative to random or greedy methods. It was shown that the GNN is suitable for modelling route networks.

3 METHODOLOGY

This section describes the implementation of the HGS based on the Local Search (LS) and the HGA. At this point, only the general functionality of these two algorithms and their interaction with each other will be discussed.

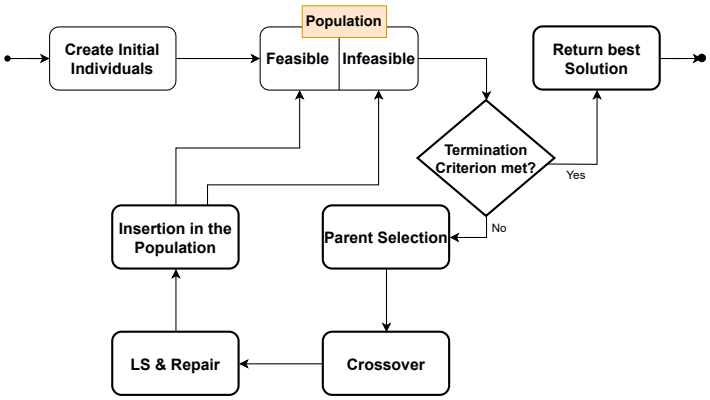

Fig. 1. Flow Chart for the sequential version of the HGS Algorithm

Figure 1 shows the flow of the sequential algorithm. The field of Local Search (LS) distinguishes the algorithm as hybrid and thus differentiates it from a purely genetic algorithm. The main goal here is to achieve speedup in the LS and Population domains. The basis for this algorithm is again the HGS for the CVRP [9].

At the beginning, an initial population is created and divided into the two subpopulations, *feasible* and *infeasible*. A solution is called infeasible if it exceeds the capacity of a vehicle or does not respect the order of pickup and delivery points.

The iterative optimization of the algorithm then starts. In each iteration, two individuals, the parents, are selected from the current population in order to generate a new individual, the so-called offspring, by crossover. The subsequent LS attempts to improve the offspring and thus place it in the subpopulation of feasible solutions. If the offspring's solution is infeasible, there is a 50% chance that repair will occur. This is a second LS performed with more stringent parameters. After repair, the offspring is placed back into one of the two subpopulations.

After each iteration, the size of one of the subpopulations is increased and it must be checked whether the maximum size has been reached. If this is the case,

Algorithm 1 HGS_CVRPPD

```
while termination_condition is not met do
    parents = SelectParents()
    offspring = SinglePointCrossover(Parents)
    LS(offspring)
    if offspring is infeasible AND rand_50() then
        TryRepair(offspring)
    InsertIntoSubPop(pop, offspring)
    if subPopulation.size() > maxSubPopSize then
        SurvivorSelection()
    AdjustPenaltyCoefficients()
bestSol = getBestFeasibleSolution()
return bestSol
```

survivors are selected according to the *survival-of-the-fittest* paradigm, which form the new population while the remaining individuals are eliminated.

In the last step, the penalty coefficients are adjusted. This process is repeated every iteration until the termination criterion is met. This can be a time limit or the maximum number of iterations without an observable improvement. As soon as the termination criterion is reached, the current best solution from the population is delivered as the result of the algorithm.

The main parameters that can be passed to the algorithm are the populations size μ and the generations size λ. Listing 1 describes the general flow of the HGS-CVRPPD. The further subsections deal in more detail with the individual steps of the HGS-CVRPPD.

3.1 Fitness Evaluation of the Individuals

One of the most important points of a GA is the calculation of the fitness value for each individual. Based on this value, decisions such as parent selection, selection of survivors, or diversification of the current population are made.

In our case, the individual is considered for the diversification of the population and not only the objective function as the goodness value. By not diversifying the population, the iterations can converge very quickly and end up in a local minimum instead of exploring more possibilities. To prevent this problem, a Biased Fitness (BF) is introduced to evaluate each individual. For this, the following two quantities are needed: $\mathcal{F}^{\phi}_{pop}(I)$: Solution Quality Rank and $\mathcal{F}^{\phi}_{Div}(I)$: Rank on the contribution of population diversification

The diversification contribution is calculated by the *average broken-pairs distance* to the n closest individuals that are the most similar to the considered individual. The BF is calculated using the following equation:

$$f_{pop}(I) = f^{\phi}_{pop}(I) + (1 - \frac{n^{el}}{|Pop|}) \cdot f^{Div}_{Pop}(I) \tag{1}$$

This equation puts more emphasis on solution quality than on diversification. Thus, we less likely lose the elite individuals during the solution search which can happen due to the diversification offsets. By doing so, we more likely improve the solution quality by retaining the best individuals and generating fewer completely different solutions.

3.2 Selection of Parents and Crossover

The goal of an iteration is to improve the existing population. For this purpose, new solutions are generated by recombination. This process consists of two steps. In the first step, two individuals P_1 and P_2 are selected. In the next step, the crossover of P_1 and P_2 is done to generate a new individual as offspring O.

The selection of the two individuals as parents is done by a so-called *binary tournament selection* (BTS) [12]. First, the BF values of the two parents are updated by the specified formula. In the next step, two individuals are randomly selected under a uniform distribution, which can be within the feasible or infeasible population. Finally, the individual with the better fitness value is selected and the process is repeated for the second parent.

As a crossover operator, we implemented a special version of the one-point crossover, which was shown to preserve the correct order of pickup and delivery pairs.

3.3 Route Splitting

So far, the tour has been treated as a single large-scale route in our algorithm, since this is advantageous in crossover. However, for the solution of the problem, several different vehicles are intended to be used. To achieve the division of the large-scale tour into several sub-tours, which are improved in the LS, the split algorithm was used.

In the later LS, moves within the route (intra-moves) as well as moves between two sub-routes (inter-moves) must be possible. A splitting algorithm, which is commonly used for solving the capacited VRP, also serves as our basis for partitioning the route [13]. Here the problem of finding delimiters is reduced to the shortest path problem and thus allows an optimal solution.

To apply this algorithm to the CVRPPD, an adjustment must be made. Splitting the total route into sub-routes should only be allowed if a pair of pickup and delivery is split into two different routes, as it is done in the Bellman-Ford algorithm [14].

Figure 2 shows the initial situation for a total route with five pairs of pickup and delivery points. For our call bus application, we need to solve an asymmetric problem, as it can be seen for the route to point s_6, which is shorter than in the reverse direction. Such asymmetries result from one-way streets or construction sites, for example. Each node after which a delimiter can be set is also marked with OK. After the iterative execution of our algorithm, it can be seen that the following order is optimal for a split between two vehicles:

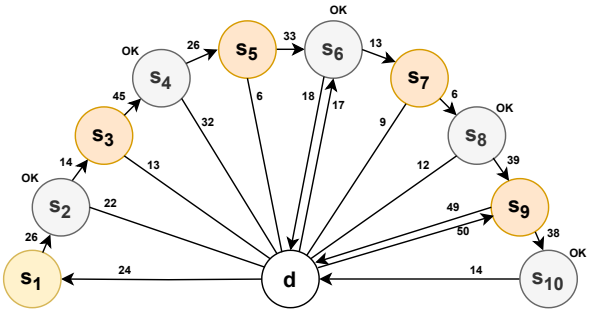

Fig. 2. Giant tour for a CVRPPD

- $vehicle1 : d \to 1 \to 6 \to 2 \to 7 \to d$
- $vehicle2 : d \to 3 \to 8 \to 4 \to 9 \to 5 \to 10 \to d$

After the split algorithm is performed, the LS is executed on the individuals.

3.4 Local Search

After the route splitting, we attempt to improve the individuals by means of LS. Here, an attempt is made to replace a solution with a better solution in its designated neighborhood. A neighborhood contains all other possible routes that can be created by predefined changes to the route, so called moves. Each neighborhood is defined by the granular search parameter γ, which limits the creation of the neighborhood to the γ nearest nodes depending on the current node, resulting in a neighborhood size of $O(\gamma \times n)$.

In general, three different categories of moves can be distinguished. These categories are *relocate-moves*, *swap-moves* and *2Opt-moves*. Each of these moves generally consists of 3 phases. First, it must be checked if there is a violation of the pickup and delivery order by the execution of the move. If this is not the case, the costs (i.e. the new distance) incurred by the move are calculated. If these are lower than the costs before the move, the move is executed [1]. In our parallel implementation, all 9 moves from the three categories are always executed simultaneously.

Especially in the LS phase, the performance of our algorithm can be optimized. On the one hand, performance optimizations can be achieved by performing precalculations and saving the results. Another optimization that we implemented was the replacement of the *first-improvement strategy* by a parallel neighborhood search: We execute all possible moves simultaneously and apply the one that leads to the greatest improvement. Thereby, we save a significant amount of time when executing the moves. Fewer moves need to be performed on an individual overall as well, since an optimum is achieved in larger increments. However, it must also be noted that a local optimum is reached more quickly due to the larger step size.

The best-improvement strategy and the first-improvement strategy are two approaches to implementing local search algorithms, especially in conjunction with metaheuristics such as simulated annealing or tabu search. Both strategies have their own advantages and disadvantages, and the choice between them depends on the specific requirements and nature of the optimization problem. The best improvement strategy looks for the best available neighborhood solution before making a decision. This means that it evaluates all possible neighborhood solutions and selects the one that improves the objective function the most. In contrast, the First-Improvement strategy uses the first neighborhood solution found that improves the objective function without checking all the others. The best-improvement strategy has the potential to find better solutions because it selects the best available neighborhood solution. Due to parallelization, the best-improvement strategy can also be applied to larger neighborhoods, since no additional effort is required to evaluate all neighborhood solutions. For convex optimization problems, such as those encountered in VRP, the best-improvement strategy helps find this minimum faster.

4 IMPLEMENTATION

In this section, our optimized implementation of the HGS using multiple processes, each responsible for a CUDA block of its own, is shown. Within the LS, we switched from the *first-improvement strategy* to a parallel *best-improvement strategy*. Further parallelization is done by executing the algorithm simultaneously on multiple individuals by splitting it among different processes. The figure also shows that a node can use multiple threads of a CPU. A GPU can be accessed by more than one node because sufficient resources are provided by the GPU. In general, a GPU on average runs the LS for 128 nodes.

Technically, the parallelization is realized by executing the algorithm simultaneously on multiple individuals, each evaluated with a different MPI process. The overall procedure is illustrated in Figure 3. The main node handles the initial creation of the population and the selection of the parents. Using MPI, we scatter the according subroutes to different processes. This is where the crossover, split algorithm, repair, and fitness evaluation for the newly generated individuals takes place.

Each of these processes in turn calls a CUDA kernel to execute all the moves of the LS simultaneously on a GPU and subsequently copies the improved solution back into the host memory according to our *best-improvement strategy*.

With the help of the MPI library, the results of the processes for the optimization of the individuals are gathered again and divided into the population according to its feasibility. If the population becomes too large, the population is adjusted based on the BF according to the *survival of the fittest* concept. Since the optimization is performed on newly created individuals, the processes do not have to wait for each other and as soon as a process has finished computing, it can receive the next individual for optimization.

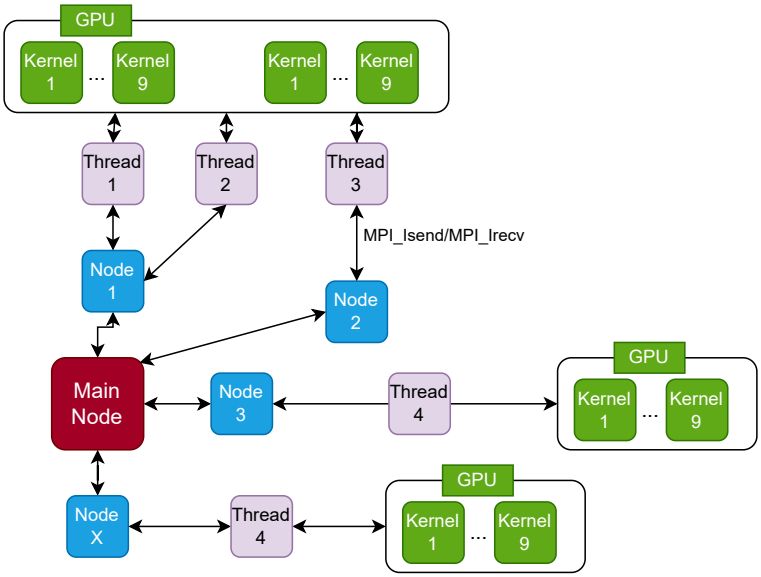

Fig. 3. Architecture of the system for the algorithm with multiple nodes and GPUs

We use *MPI_Isend* and *MPI_Irecv* for exchanging the results from the CUDA-blocks asynchronously, i. e. all the communication in our parallel CVRPPD-solver is non-blocking. We allow population accesses to overlap, thus avoiding synchronization-related waiting times which further inceases the efficiency of our implementation.

Since the MPI library enables the communication via messages exchange between processes with distributed memory, we can use it complementary to CUDA, which enables the parallelization on a single node. We benefit from the combination of the two technologies in multiple ways: Our system can cope with route optimization problems comprised of so many stops that they do not fit in the memory of a single GPU. Particularly, in large-scale route optimization, as they occur in practice when routes conncet distinct rural areas, computations can also be disproportioanly long. Using a CUDA-aware MPI implementation, we gather partial results using GPU-to-GPU commincation [15].

A CUDA kernel is used to implement the parallel execution of all moves. Since each move should be executed by its own thread, a kernel with a $1x1x9$ grid can be used. The variable *threadIdx.x* is used to assign one of the nine possible moves to each thread. The return value is both the value obtained by an improvement and a pointer to the modified individual. Since the size of an individual cannot be changed by the LS, the size is already known and there is no need for dynamic allocation of memory. Back in the initiating process, the result is copied to the host memory.

Here, the best result of all moves is determined and compared to the initial solution. The greatest improvement is further used in the next phase of our

algorithm. If all moves produce a worse solution than the initial solution, it is used for further computation.

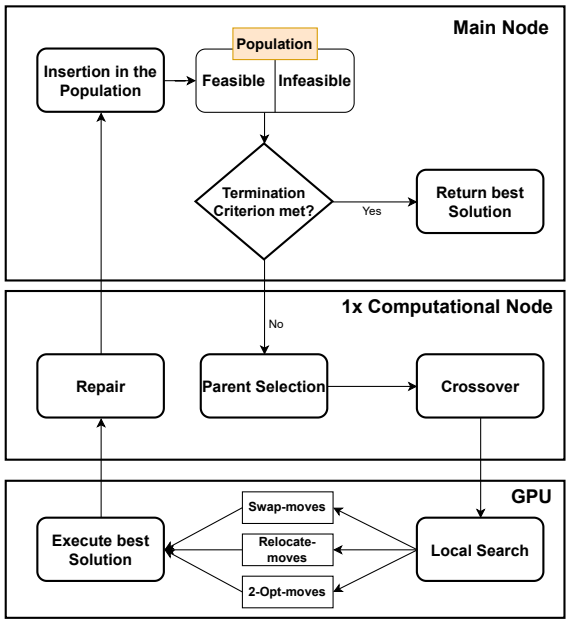

Fig. 4. Flowchart for the parallelized algorithm

Figure 4 illustrates our parallel workflow. The steps enclosed in square boxes are each executed in parallel. Our parallelization made it possible to work with multiple individuals from the population at the same time, instead of sequentially handling one individual at a time. This is possible mainly due to the possibility of asynchronous communication. Only the execution of the survival of the fittest operation and the creation of the initial population are inherently sequential. By splitting moves among CUDA threads, a further optimization was gained saving time within the CUDA blocks.

When combining code for parallelization with CUDA and MPI, some pitfalls must be considered. Due to the different runtimes and calling conventions, we needed to link C++ code (CUDA) and C code together. We split the iterations of the genetitc algorithm and the LS, such that we could keep all CUDA and all MPI code independently in different files. Using the respective compilers (`mpicc` & `nvcc`), all sources (`*.c` & `*.cu`) are first compiled to object files. Afterwards, the linking and the creation of the executable is done with the help of *mpicc* linking the binaries against the CUDA library. To simplify this process, an automated build was implemented using a makefile. The resulting executable file is then run

using *mpiexec*, configured for using the desired resources (processes and GPUs) in a submission script passed to the SLURM scheduler used at EPCC [16].

5 EVALUATION AND RESULTS

The algorithm was tested for 4 benchmark problems selected by us. These problems differ in the number of pickup and delivery pairs that are to be approached within a route. The number of nodes ranges from 50-10000 nodes.Here, the 50-pair problem is roughly equivalent to the normal deployment within a CVRPPD. In our special case, where mainly rural areas are considered, the problem size can increase significantly. This happens on the one hand due to the long travel distances with simultaneously high demand, whereby vehicles never become completely empty and no calculation of a fresh tour can take place. A problem size of 200 is therefore quite realistic for our use case. The two other problem sizes were chosen to explore the limits of our parallelization. A problem size of $10,000$ pairs can at most take place in the case of mail. However, this is reduced by collection centers and so the problem size decreases here as well. The time for execution is given in performed generations (i.e. number of optimizations) per second. A number of 500 iterations without improvement was chosen as termination criterion. In addition, the total time needed to solve the problem is also given. The same number of 9 threads is always used for the distribution to the CUDA threads.

Table 1. Runtimes and Speedup in seconds for selected test instances

# of Nodes	Seq.	16	128	256	1080
Runtime in s					
50 Pairs	1.937	2.381	2.595	2.764	3.181
200 Pairs	3.371	3.294	3.245	3.183	3.292
1,000 Pairs	34.07	27.18	25.19	22.18	20.22
10,000 Pairs	180.7	120.3	95.4	78.8	39.2
Speedup					
50 Pairs	1x	0,81x	0,75x	0,70x	0,61x
200 Pairs	1x	1,02x	1,04x	1,06x	1,02x
1,000 Pairs	1x	1,25x	1,35x	1,54x	1,69x%
10,000 Pairs	1x	1,50x	1,89x	2,29x	4,61x

Table 1 shows the average runtime in *ms* for a parallelization with 16, 128, 256 and 1080 threads. In addition, the duration for the runtime of the sequential algorithm is also given and the runtime of the open-source algorithm VROOM, based on a tabu search for the solution of the CVRPPD [17].

The systems used for these calculations were run on a computer with two AMD EPYC 64-core processors. The 1080 thread tests used the Cirrus UK National Tier-2 HPC Service at EPCC [3]. Based on the table, we can see that the runtime of the parallel algorithm is up to 1.6 times slower than the sequential execution on a single thread due to the increased communication overhead for

the smallest instance. From a problem size of 200 pairs, the parallelization could achieve better results and the solution of the large-scale problem with 10,000 pairs our parallelization reaches a speedup of almost 80%. Thus, as the number of nodes increases, an optimal result can be achieved more quickly. It should be noted, however, that the improvement in runtime is not linear with the number of nodes, but decreases as the number of nodes increases.

The closeness of our solutions to the results of a local search with *first-improvement strategy* are a first promising indication that our solutions are very close to the global optimum. In addition, a comparison was made with the global optimum of solutions, which was determined by us for certain problem instances by trying out all possible solutions. Here, our algorithm achieved an average deviation of 1.29% from the optimal solution. The selected problem instances had a size between 20 and 100 pairs, because it would be impossible to calculate an optimal solution for larger problems.

The proven algorithm has so far been applied as a case study for the Roding transport authority and has achieved impressive results there. This suggests that this algorithm has the potential to be applied to other problems as well. In the specific use case, the algorithm achieved good results, but the parallelization could not yet be fully exploited. By transferring it to other regions, such as large cities with many more necessary stopping points, new insights could be gained, efficiencies could be improved, or further innovative solutions could be found. This approach allows our case study to be used as a basis for solving more diverse, similar challenges and to fully exploit the potential of the algorithm.

6 CONCLUSION

This work uses an HPC cluster comprising multiple GPU servers for solving a real-world problem that directly impacts public transit optimization. A parallel genetic algorithm was presented along with early and parallel execution of local search. The algorithm was tested on 2 different multi-GPU systems with NVIDIA A100 GPUs to solve large-scale CVRPPD with up to $10,000$ nodes. To take advantage of maximum parallelization, an approach of trading one individual from the population of one thread each was chosen. It was found that parallelization produces significant improvements in runtime for problem sizes as small as $2,000$ bus stops compared to sequential execution on a CPU or without the parallelization provided by CUDA. In our application, these $2,000$ stops are actually necessary because in rural areas the stops are often far apart and a large area needs to be covered. To cover this space accordingly with new stops and to minimize the walking distances of the population, $2,000$ of these pop-up stops were introduced.

References

1. T. Stadler, S. Hofmeister, and J. Dünnweber, "A method for the optimized placement of bus stops based on voronoi diagrams," in *Proceedings of the Annual Hawaii*

International Conference on System Sciences, Hawaii International Conference on System Sciences.
2. I. Lana, J. D. Ser, M. Velez, and E. I. Vlahogianni, "Road traffic forecasting: Recent advances and new challenges," vol. 10, no. 2, pp. 93–109.
3. A. Turner, "Cirrus user guide." https://cirrus.readthedocs.io/en/main/user-guide/introduction.html. last accessed on: 01.09.2023.
4. T. Pichpibul and R. Kawtummachai, "A heuristic approach based on clarke-wright algorithm for open vehicle routing problem," vol. 2013, pp. 1–11.
5. B. D. Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser, "Solving vehicle routing problems using constraint programming and metaheuristics," vol. 6, no. 4, pp. 501–523.
6. S. Hoffmann and R. Lienhart, *OpenMP: Eine Einführung in die parallele Programmierung mit C/C++*. Springer-Verlag, 2008.
7. D. Kirk *et al.*, "Nvidia cuda software and gpu parallel computing architecture," in *ISMM*, vol. 7, pp. 103–104, 2007.
8. P. Yelmewad and B. Talawar, "Parallel version of local search heuristic algorithm to solve capacitated vehicle routing problem," vol. 24, no. 4, pp. 3671–3692.
9. T. Vidal, "Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood," vol. 140, p. 105643, apr 2020.
10. M. Abdelatti, M. Sodhi, and R. Sendag, "A multi-gpu parallel genetic algorithm for large-scale vehicle routing problems," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2022.
11. I. Santana, A. Lodi, and T. Vidal, "Neural networks for local search and crossover in vehicle routing: A possible overkill?." https://arxiv.org/abs/2210.12075, 2022.
12. G. Homsi, R. Martinelli, T. Vidal, and K. Fagerholt, "Industrial and tramp ship routing problems: Closing the gap for real-scale instances," vol. 283, pp. 972–990, jun 2018.
13. C. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem," vol. 31, no. 12, pp. 1985–2002.
14. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press.
15. R. Saxena, M. Jain, S. Bhadri, and S. Khemka, "Parallelizing GA based heuristic approach for TSP over CUDA and OPENMP," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE.
16. M. A. J. Andy B. Yoo and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proceedings of the ClusterWorld Conference and Expo*, Springer LNCS.
17. J. Coupey, "New features for our route optimization api." https://blog.verso-optim.com/2022/05/31/solving-problems-better-and-faster/. last accessed on: 01.09.2023.