Applied Intelligence
https://doi.org/10.1007/s10489-018-1210-6

GMMA: GPU-based multiobjective memetic algorithms for vehicle routing problem with route balancing

Zizhen Zhang[1] · **Yuyan Sun**[1] · **Hong Xie**[1] · **Yi Teng**[2] · **Jiahai Wang**[1]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract

A multiobjective optimization problem called a vehicle routing problem with route balancing (VRPRB) is studied. VRPRB extends traditional VRPs by considering two objectives simultaneously. The first objective is the minimization of the total traveling cost and the second one tries to ensure the balance among multiple routes. Different from another commonly used balancing objective, namely, the minimization of the difference between the maximal and minimal route cost, the objective we introduce is the minimization of the maximal route cost. Such setting can effectively avoid the occurrence of distorted solutions. In order to find Pareto-optimal solutions of VRPRB, we develop a multiobjective memetic algorithm (MMA), which integrates a problem-specific local search procedure into a multiobjective evolutionary algorithm. The MMA is further enhanced by using parallel computations on GPU devices. A simple version and a revised version of GPU-based MMAs are proposed and implemented on the CUDA platform. All the algorithms are tested on the benchmark instances to demonstrate their efficacy and effectiveness. Furthermore, the performances of CPU-based and GPU-based algorithms are analyzed.

Keywords Vehicle routing problem · Multiobjective · Parallel · GPU · CUDA

1 Introduction

In this work, we investigate a vehicle routing problem with route balancing (VRPRB) and devise highly parallelized algorithms for solving it. VRPRB is an extension of the well-known VRP [28], which is considered as one of the most important topics in the scope of combinatorial optimization, transportation, operations research and so on. VRPs describe real-world applications concerning the delivery (and/or the pick-up) of goods to customers. The optimal set of routes has to be determined subject to various kinds of constraints.

Traditionally, the studies on VRP and its extensions mostly aim at optimizing a single target, e.g., the operational cost or the incurred traveling time. In the last two decades, the research on multiobjective optimization has been receiving more and more attention. This is because 1) the presence of multiple objectives is natural in many applications [6], and 2) the maturity of evolutionary algorithms and other heuristic approaches provides standard and accessible frameworks for optimizing multiple objectives simultaneously.

VRPRB we study is a multiobjective optimization problem. Its solution space is the same as that of the classic capacitated VRP, which is detailed in Section 3. And the following two objectives are taken into consideration.

1. Minimization of the total traveling cost.
2. Minimization of the maximal route cost.

The first objective of VRPRB is originated from the classic VRP and the second one is regarded as the route balancing objective. It is worth noting that some approaches use the minimization of the difference between the maximal and minimal route cost for route balancing (e.g., [12]). We point out in Section 3 that such setting may result in *distorted* solutions, but our setting guarantees to produce doable solutions.

In order to solve multiobjective optimization problems, the *posteriori* method [6] is commonly adopted, i.e., *Pareto-optimal* solutions are generated and then the decision-maker makes a trade-off between different objectives and selects the solution(s) based on his preference or experience. A

✉ Zizhen Zhang
zhangzizhen@gmail.com

[1] School of Data and Computer Science, Sun Yat-sen University, Guangdong Province, China

[2] Guangdong Information Technology Bureau, China Post Group Co., Ltd., Guangdong Province, China

Published online: 05 June 2018

solution is called Pareto-optimal, if it is *non-dominated* by other solutions in the solution space. Pareto-optimal solutions constitute a *Pareto front*. In practice, finding the Pareto front of a complex problem such as VRPRB is computationally intractable. A conventional approach is to obtain a set of non-dominated solutions to approximate it. To do so, applying multiobjective evolutionary algorithms may be a desirable option. In addition, local search is one of the critical components in meta-heuristics for solving discrete optimization problems. Thus, we decide to integrate problem-specific local search operators, which basically use the concept of *memetic* algorithms, into an evolutionary framework. We formally call our approach a *multiobjective memetic algorithm* (MMA).

The proposed MMA can attain satisfactory results for VRPRB according to our experiments. However, there is still room to improve the MMA in terms of running time. This can be done by parallelizing the algorithm and executing with multiple devices. Recently, along with the development of artificial intelligence techniques, GPU (graphics processing unit) programming has been witnessed its success in many computing areas, e.g., computer games, image processing, neural computing, etc. A GPU is a highly parallel, multithreaded, manycore processor with tremendous computational horsepower and very high memory bandwidth [10]. GPUs are available in most personal computers and originally designed to render computer graphics which are to be displayed on the screen. Nowadays, in addition to traditional graphics-rendering tasks, GPU programs can also address general-purpose tasks [24]. To apply GPU computing in a convenient way, a widely adopted programming model is the NVIDIA's CUDA (compute unified device architecture) platform. It allows programmers to program with Nvidia GPUs to perform general-purpose computing tasks. It leverages the parallel computer engine to solve many complex computational problems in an efficient way [10]. Therefore, our work tries to transplant the execution of the MMA to GPU by using the CUDA platform.

The remaining of the paper is organized as follows. Section 2 gives some related work on VRPRB and GPU programming. In Section 3, we present a formal definition and some properties of VRPRB. In Section 4, we describe an MMA for solving VRPRB. It is then extended to two GPU-based parallel versions, which is detailed in Section 5. In Section 6, we report a series of experiment results to demonstrate the effectiveness and efficiency of our algorithms. Finally, conclusions are drawn in Section 7.

2 Related work

We first review the literature related to VRPRB. [12] proposed an evolutionary algorithm for VRPRB, in which two objectives are defined by the minimization of total route length and the minimization of difference between the maximal and minimal route length. An elitist diversification mechanism and a parallel model were used within their approach. With the same balancing objective, [14] proposed an algorithm called Multi-Start Split-based Path Relinking for VRPRB and compared their results with [12]. [16] also considered the same balancing objective and designed a memetic NSGA-II algorithm to solve the problem. [15] studied the VRP with load balancing. They developed a heuristic algorithm to minimize the travel length and balance drivers' load simultaneously. The proposed balancing objective is to minimize the sum of working time difference between every vehicle and the vehicle with the smallest working time. [17] introduced an algorithm based on the scatter search metaheuristic to solve a real problem from a company in Tenerife, Spain. Daily routes have to be designed for a given fleet of vehicles so as to minimize the total traveling distance while balancing the workload of drivers. The balancing objective is to minimize the difference between the maximum and minimum route length. [13] suggested a new approach based on the free disposal hull to solve the VRP with time windows (VRPTW). Apart from minimizing the number of vehicles and the total traveling distance, they also considered the balance of the load carried by each active member of the vehicle fleet. [9] treated VRPTW as a bi-objective problem, which tried to minimize the number of routes and the total cost simultaneously. They focused on the similarity of solutions in the recombination process and a method based on Jaccard's similarity was adopted. [21] devised a novel algorithm, which combined a local search and a variant of a greedy randomized adaptive search procedure, to find VRPRB solutions. They tried to achieve route balancing by minimizing the difference between the longest and the shortest route. [31] proposed a genetic algorithm with a complex chromosome representation to solve a bi-objective VRPTW, which considered the total traveling distance as well as the workload imbalance of vehicles. To better address the balancing issue, they designed a balance factor, which is the difference between the maximal and minimal route length divided by the average route length. [22] conducted research on a bus routing problem. They formulated the problem into a bi-objective optimization model that dealt with the minimization of both the longest route length and the total route length. Tabu search within a multiobjective adaptive memory programming framework was proposed to solve the problem. [26] studied VRPRB and developed an NSGA-II based memetic algorithm. The experiments showed that their results are superior to those in [12].

Next, the literature on GPU-based algorithms for related routing problems is presented as follow. [4] gave a brief introduction to modern PC architectures and GPU programming. To illustrate the execution model of GPU, a simple example of a GPU-based local search procedure was presented. Moreover, strategies and guidelines for software development and performance optimization were discussed. [25] conducted a comprehensive survey on GPU computing in the area of discrete optimization. It showed that GPU computing in discrete optimization, especially in routing problems, is still in its infancy. But in later years, more and more routing problems were tackled with GPU computing. For example, [30] proposed a multi-GPU parallel memetic algorithm for the capacitated VRP. They used a hierarchical Parallel Random Access Machine (PRAM) model to analyze the time complexity and cost of the algorithm. The algorithm was coded in C++ using CUDA and OpenMPI library. [27] used parallel tabu search algorithm to solve a multi-criteria distance-constrained VRP. In each iteration of the algorithm, parallel neighborhood search is performed on the solutions from current Pareto frontier. To solve the VRP with single and multi-depots, [2] proposed a GPU implementation of a heuristic method based on Clarke and Wright algorithm. The proposed algorithm generates an initial solution in parallel in one step and then iteratively improves the costs of all pairs of neighbor vehicle routes, also in parallel. Experimental results demonstrated that the proposed implementation exploited the parallelism of GPU efficiently. [24] studied the implementation of local search for the vehicle routing problem on GPU. The local search method they considered was the best improving strategy for 2-opt and 3-opt operators using the giant tour representation. The experiments demonstrate that local search can be implemented in an efficient way on GPU. [29] established a new guideline to design and implement Local Search Metaheuristics (LSMs) effectively on GPU. In this paper, efficient approaches were proposed for the mapping of neighboring solutions to GPU threads. They also proved the effectiveness of the proposed method through four well-known combinatorial and continuous optimization problems. [8] investigated a parallelization of 2-opt and 3-opt local search heuristics for the traveling salesman problem. The insight of parallelization strategies for better utilizing GPU resources was discussed. [23] developed a multi-GPU and multi-core metaheuristic based on the iterated local search. Experimental results for an NP-hard problem (i.e., minimum latency problem) demonstrated the effectiveness of the proposed techniques in terms of solution quality, performance and scalability.

3 Problem definition, formulation and properties

VRPRB is defined on a complete and undirected graph $G = (N, E)$, where $N = \{0, 1, \ldots, n\}$ is the node set and $E = \{(i, j) | i, j \in N\}$ is the edge set. Node 0 indicates the depot and nodes $1, \ldots, n$ correspond to n customers. Define by c_{ij} the traveling cost between node i and node j, and define by d_i the demand of each customer $i \in N \setminus \{0\}$. For convenience, we assume that $d_0 = 0$. Let V be the collection of m homogeneous vehicles and C be the capacity of a vehicle. VRPRB requires that each customer must be serviced by exactly one vehicle, and the load of a vehicle must not exceed its capacity.

VRPRB is a bi-objective problem and we can provide a mathematical model to formulate it. Three sets of decision variables are introduced as follows.

x_{ijk}: a binary decision variable which represents the number of times that vehicle k traverses edge (i, j);

y_{ik}: a binary decision variable which is equal to 1 if customer i is serviced by vehicle k, and 0 otherwise;

u_{ik}: a continuous variable which denotes the load of vehicle k after servicing customer i.

The bi-objective optimization model is:

$$\min\{f_1(x, y, u), f_2(x, y, u)\} \tag{1}$$

subject to

$$\sum_{k \in V} y_{ik} = 1, \ \forall\, i \in N \setminus \{0\} \tag{2}$$

$$\sum_{k \in V} y_{0k} = m \tag{3}$$

$$\sum_{j \in N} x_{ijk} = \sum_{j \in N} x_{jik} = y_{ik}, \ \forall\, i \in N, \ k \in V \tag{4}$$

$$\sum_{i \in N} d_i y_{ik} \leqslant C, \ \forall\, k \in V \tag{5}$$

$$\begin{aligned} u_{ik} - u_{jk} + C x_{ijk} \leqslant C - d_j, \ \forall\, i, \ j \in N \setminus \{0\}, \\ i \neq j, \ k \in V, \ d_i + d_j \leqslant C \end{aligned} \tag{6}$$

$$d_i \leqslant u_{ik} \leqslant C, \ \forall\, i \in N \setminus \{0\}, \ k \in V \tag{7}$$

$$y_{ik} \in \{0, 1\}, \ \forall\, i \in N, \ k \in V \tag{8}$$

$$x_{ijk} \in \{0, 1\}, \ \forall\, i, \ j \in N, \ k \in V \tag{9}$$

In the model, Constraints (2) ensure that each customer must be serviced by exactly one vehicle. Constraints (3) guarantee that all vehicles must start from the depot. Constraints (4) are the flow conservation constraints. Constraints (5) are the capacity constraints. Constraints (6) and (7) are subtour elimination constraints, which were originally proposed for the traveling salesman problem (TSP) by Miller et al [19] and later modified to suit the VRP.

The objective (1) consists of two functions. The first function is to minimize the total traveling cost:

$$f_1(x, y, u) = \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{10}$$

The second function is to optimize the balance among all the vehicle routes, which can be written as:

$$f_2(x, y, u) = \max_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{11}$$

We call the minimization of $f_2(x, y, u)$ a *min_max* method for route balancing. It is worth pointing out that another commonly used objective function is to minimize the difference between the maximal and minimal route cost, namely,

$$\bar{f_2}(x, y, u) = \max_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} - \min_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{12}$$

However, minimizing $\bar{f_2}(x, y, u)$ may result in the occurrence of *distorted* solutions. A solution s is called *distorted*, if s is Pareto-optimal but some route(s) is not optimal in terms of the total traveling cost.

Figure 1 illustrates an example of the distorted solution. It contains two vehicle routes marked in solid lines. The square and the circles represent the depot and the customers, respectively. Routes 1 and 2 have the same total traveling cost, which is equal to 6. The difference between the maximal and the minimal route cost is 0. When the second objective is given by $\bar{f_2}(x, y, u)$, two objectives of this solution can be represented by (12, 0). If we replace Route 2 with the route indicated by the dash lines, then the cost of Route 2 becomes 5 and the two objectives of the solution become (11, 1). Both of the solutions are Pareto-optimal. However, it is obvious that the previous one is *distorted* because Route 2 takes a detor to achieve the balance.

When the second objective is defined by $f_2(x, y, u)$, no matter Route 2 is constituted by the solid lines (the objectives can be represented by (12, 6)) or the dash lines (the objectives can be represented by (11, 6)), the corresponding solutions have the same value of f_2. Therefore, (11, 6) is the only Pareto-optimal solution. The following theorem ensures the effectiveness of our objective settings in preventing distorted solutions.

Theorem 1 *For a Pareto-optimal solution* $s = (x, y, u)$ *with two objectives* $f_1(s)$ *and* $f_2(s)$ *defined by Expression* (10) *and* (11)*, each route in solution* s *must be an optimal TSP tour.*

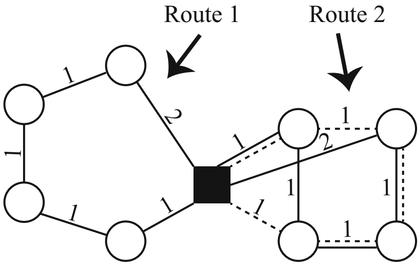


Fig. 1 An example of the distorted solution

Proof Proof by contradiction. Assume that s is Pareto-optimal with some route r which is not TSP optimal. Then route r can be improved into a TSP optimal route r', leading to a new solution s'. Because the cost of route r' is smaller than that of route r, $f_1(s') < f_1(s)$. For the second objective, two cases may happen.

Case 1. Route r has the maximal route cost in solution s. If route r is replaced with route r', the maximal route cost of solution s' must be given by route r' or the other route shorter than route r. Hence, $f_2(s') < f_2(s)$ and solution s is dominated by s'.

Case 2. The maximal route cost in solution s is not given by route r. If route r is replaced with route r', the maximal route cost of solution s' remains unchanged. Therefore, solution s is also dominated by s' due to $f_1(s') < f_1(s)$ and $f_2(s') = f_2(s)$.

To sum up, either of the above two cases contradicts with the assumption that s is Pareto-optimal, which proves the correctness of the theorem. ☐

4 CPU-based multiobjective memetic algorithm

In this section, we describe a CPU-based multiobjective memetic algorithm (CMMA) for approximating Pareto-optimal solutions of VRPRB. The CMMA is essentially a modification of the well-known multiobjective algorithm NSGA-II [7]. It also borrows the concept of memetic algorithm [20], in which a domain-specific refinement is applied to improve the solution quality locally.

4.1 Solution framework

The framework of the proposed CMMA is shown in Algorithm 1. In line 1, an initial population P consisting of $|P|$ individual solutions is produced. Line 2 – 13 are the evolutionary process. The evolution stops when the number of generations exceeds N_{gen}. In line 3, a **recombination** procedure is employed to generate a new population P'. Next, $Iter$ iterations are performed. In each iteration, a guiding vector w is first generated (line 5). Subsequently, a solution s is chosen from $P \cup P'$ according to the **tournament selection** (line 6). Then local refinements are conducted LS_iter times to improve the quality of solution s. One of K local search (**LS**) operators is randomly chosen and applied to solution s (line 8 – 9). In line 12, an **update** procedure is involved to form a new population for the next generation.

Detailed solution procedures of Algorithm 1 are provided in the next few subsections. From this algorithm, we can observe that the LS operators are invoked $N_{gen} \cdot Iter \cdot LS_{iter}$ times in total. All the codes are serially executed with a CPU core.

Algorithm 1 The framework of the CMMA for solving VRPRB

```
Generate an initial population $P$;
while the number of generations does not exceed $N_{gen}$ do
    $P' = $ recombination$(P)$;
    for $i = 1$ to $Iter$ do
        Generate a weight $w = (w_1, w_2)$;
        $s \leftarrow$ tournament_selection$(P \cup P', w)$; // select a solution for local improvement
        for $l = 1$ to $LS_{iter}$ do
            $k \leftarrow$ generate a random integer in the range $[1, K]$;
            $s \leftarrow$ LS$_k(s, w)$; // use the $k$-th LS operator
        end for
    end for
    $P = $ update$(P, P')$;
end while
return $P$;
```

4.2 Solution encoding and decoding

The solution space of VRPRB is exactly the same as that of the traditional VRP. A solution consists of m routes, where each route can be represented by a customer visiting sequence. Because a customer is visited exactly once, we can also use a *giant tour* [1] to encode a solution. A giant tour is a tour starting from the depot and visiting all the customers. It is essentially a permutation of the node set N where 0 (the depot) is always the first node in the permutation. The giant tour representation can be very helpful in the initialization phase and the recombination phase.

To decode a giant tour, we have to split it into a set of m feasible vehicle routes. Because there are different ways for the splitting, and VRPRB has two objectives, we would like to decide the optimal splitting when only one objective is taken into consideration.

Formally, suppose that a giant tour is denoted as $(0, \pi_1, \pi_2, \ldots, \pi_n)$, where each element corresponds to a customer index. The first splitting method aims at seeking a splitting such that the total traveling cost is minimized. Let the state $T[k][i]$ be the minimum traveling cost for the first k vehicles sequentially visiting customers $(\pi_1, \pi_2, \ldots, \pi_i)$. The dynamic programming recursion for calculating $T[k][i]$ is:

$$T[k][i] = \min_{0 \leqslant j \leqslant i \text{ and } cap(j+1,i) \leqslant C} \{T[k-1][j] + cost(j+1, i)\} \quad (13)$$

where $cost(j, i)$ and $cap(j, i)$ correspond to the traveling cost and the required capacity of route $(0, \pi_j, \ldots, \pi_i, 0)$, that is,

$$cost(j,i) = \begin{cases} c_{0,\pi_j} + c_{\pi_i,0} + \sum_{j'=j}^{i-1} c_{\pi_{j'},\pi_{j'+1}}, & \text{if} j \leq i; \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

$$cap(j,i) = \begin{cases} \sum_{j'=j}^{i} d_{\pi_{j'}}, & \text{if} j \leq i; \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

The boundary condition is $T[0][0] = 0$ and the optimal splitting value is $T[m][n]$, i.e., the minimum total traveling cost achieved by dispatching m vehicles to service all customers.

The second splitting method tries to find a splitting that can minimize the maximal route cost. Define by $F[k][i]$ the minimum value of the maximal route cost for the first k vehicles visiting customers $(\pi_1, \pi_2, \ldots, \pi_i)$. Similar to (13), the calculation of $B[k][i]$ is given by:

$$B[k][i] = \min_{0 \leqslant j \leqslant i \text{ and } cap(j+1,i) \leqslant C} \{\max\{F[k-1][j], cost(j+1, i)\}\} \quad (16)$$

The initial state is $B[0][0] = 0$ and the final optimal value is $B[m][n]$.

The above descriptions show that, given a giant tour, it can be decoded into two VRPRB solutions according to different splitting methods. The time complexity of each splitting method is $O(mn^2)$.

4.3 Initial population

The initial population contains $|P|$ solutions, which can be simply determined as follows. First, $|P|/2$ permutations are randomly generated to form giant tours. Next, the two splitting methods are applied on each giant tour to obtain a total of $|P|$ solutions.

4.4 Recombination

The recombination procedure tries to produce a new population P' in which the solutions can inherit some characteristics of solutions in the original population P. The recombination first encodes all the solutions into giant tours. Next, it randomly mates giant tours in pairwise. For each pair, the two giant tours are regarded as parents and they are recombined to generate one offspring giant tour using the *order crossover* (OX) operator. Finally, for each resultant offspring giant tour, the two splitting methods are used to generate two VRPRB solutions and then they are added to the population set P'. Finally, we must have $|P'| = |P|$ offspring solutions.

4.5 Local search operators

Local search (LS) operators are very important in designing an effective meta-heuristic. We devise the following four (i.e., $K = 4$ in line 8 of Algorithm 1) operators in LS procedure. These operators are also widely used in other meta-heuristics for VRPs (e.g., [3]).

1. $exchange(i, j)$. Customer i in route r_1 is replaced by customer j in route r_2. Customer j in route r_2 is replaced by customer i in route r_1.
2. $relocate(i, j)$. Customer i in route r_1 is removed and inserted into the position after customer j in route r_2.
3. $2 - opt(i, j)$. All the customers after position i in route r_1 are removed and inserted into the position after customer j in route r_2. All the customers after position j in route r_2 are removed and inserted into the position after customer i in route r_1.
4. $reverse(i, j)$. Different from the above operators, the reverse operator is used for a single route. Customers from position i to position j are reversed, changing a part of the route from $(i, i + 1, \ldots, j - 1, j)$ to $(j, j - 1, \ldots, i + 1, i)$.

For each operator, two different customers i and j are randomly selected from solution s. After performing the operator, the value of the resultant solution is measured by the weighted sum of two objectives, where the weights are given according to the guiding vector w. If the weighted sum decreases, the corresponding operator is accepted and solution s gets improved. Note that all the operators must guarantee the feasibility of their resultant solutions.

4.6 Tournament selection and population updating

At each iteration, a solution s is selected from the population $P \cup P'$ for the local refinement. We use the tournament selection method [18] to determine the solution. Specifically, a percentage τ of individual solutions are first chosen at random from $P \cup P'$. The winner individual (the one with the smallest weighted sum) is then selected.

For the population updating procedure, the standard non-dominated sorting and crowding-distance approaches are employed. These methods are originated from NSGA-II. For more details of population updating, we refer the readers to [7].

5 GPU-based multiobjective memetic algorithm

The proposed CMMA suffices to yield excellent results according to the computational experiments (see Section 6). However, it is still not efficient enough, because it maintains a large population and involves many iterations. After carefully analyzing the framework of CMMA, we find that many procedures can be parallelized for accelerating the execution.

In this section, we describe two GPU-based multiobjective memetic algorithms (GMMAs): one slightly modifies CMMA with transferring code snippets on GPU; the other re-designs the algorithm framework and extensively exploits the power of GPU. To start with, the architecture of GPU and the concept of CUDA programming are introduced.

5.1 GPU hardware and CUDA programming

Figure 2 shows the hardware structure of a GPU, which consists of an array of Streaming Multiprocessors (SMs) [10]. Different GPUs may have different numbers of SMs. An SM has its own memory and many simple processors that can run a bunch of parallel threads. When a multithreaded program is executed, GPU is responsible for assigning thread blocks to hardware SMs. All the SMs run independently and in parallel.

In order to utilize the GPU hardware, several important concepts of the CUDA programming model are stated as follows.

1. *Thread, Block and Grid*. *Thread* is the smallest unit in GPU that can be executed [2]. CUDA works in terms of threads. They execute independently of each other unless explicitly synchronized. Threads which cooperate to solve a problem can be grouped into a *block*. All threads in a block execute the same code on the same SM at the same time. Blocks are arranged into a *grid*.
2. *Kernel*. A *kernel* is a C function qualified with the *__global__* keyword. Regular C functions are executed only once with one call. However, kernels are executed N times in parallel on GPU, by N different CUDA threads according to kernel launch parameters. A typical kernel call is as follows:

   $KERNEL <<< dim3(x_1, y_1, z_1), dim3(x_2, y_2, z_2) >>> ();$

   The function name of the kernel call is *KERNEL*. The launch parameters of this kernel are $dim3(x_1, y_1, z_1)$

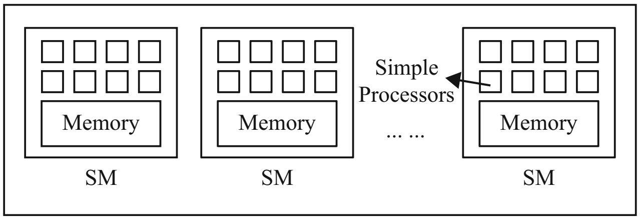
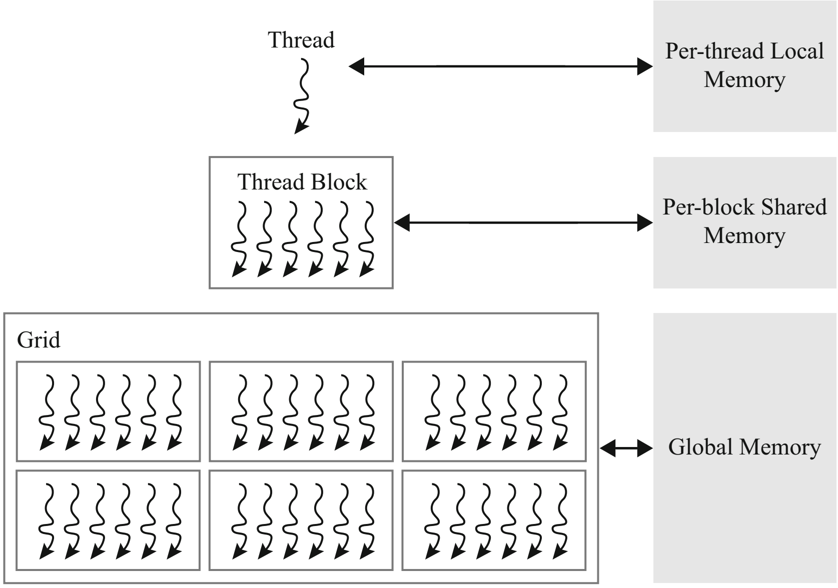


Fig. 2 The GPU hardware structure

Fig. 3 Memory Hierarchy



and $dim3(x_2, y_2, z_2)$, which specify the dimensionality of the grid of blocks and the dimensionality of the threads within a block, respectively. GPU is capable of running many blocks at the same time. Each block has a maximum number of threads that it can support. Each thread can be uniquely indexed by a thread index and a block index which are provided inside the code.

3. *Memory Hierarchy*. When CUDA threads are executed, they may access data from multiple memory spaces as illustrated by Fig. 3. Each thread has access to local memory which is private to that thread. Threads can read and write from local memory. Each thread block has shared memory visible to all threads in the block. The threads in the block can communicate with each other through reading and writing to per-block shared memory. All threads in the entire system have access to global memory.

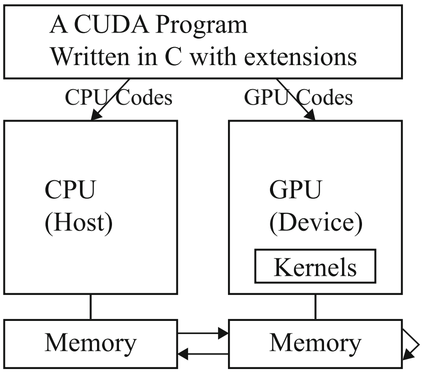


Fig. 4 CUDA programming diagram

The CUDA programming model makes it possible to integrate both CPU codes and GPU codes within a single program. Figure 4 shows the CUDA programming diagram. The plain codes of the program are run on CPU, while the codes with CUDA qualifiers are run on GPU in parallel. The program will be split into pieces for CPU and GPU by the CUDA compiler. The memory from CPU (host) and from GPU (device) can exchange with each other using the *cudaMemcpy* function.

5.2 A simple GPU-based multiobjective memetic algorithm

A straightforward way to parallelize CMMA by the GPU platform is to use the so-called *algorithmic-level parallel model* [29], where local search operators are performed independently and in parallel on the solutions inside the population. A simple version of GPU-based multiobjective memetic algorithm (S-GMMA for short) is proposed and the framework is shown in Algorithm 2.

At the beginning (line 1), an initial population P_h is generated, where the subscript "h" stands for "host" (i.e., CPU side). This line means that the population P_h is stored in the host memory. In each generation, the recombination is conducted on P_h, resulting in an offspring population P'_h. Next, in order to parallelize the local search, the tournament selection process adopted in CMMA is removed. Instead, all the offspring solutions in P'_h are chosen for the local refinement. To do so, P'_h is copied to P_d using *cudaMemcpy*, where the subscript "d" in P_d stands for "device" (i.e., GPU side). Thereafter, local search is performed on GPU: kernel calls are launched for processing P_d (line 6). The kernel function is **S-GMMA-LS**. The launch parameters $dim3(1)$

and $dim3(x_2^s, y_2^s)$ respectively mean that the number of blocks is 1 and the number of threads is $x_2^s \cdot y_2^s$ (the total number of threads must no less than $|P|$). Once the kernel calls are finished, P_d is copied back from device to host (line 7). The updating method of the population is the same as that in CMMA. When the number of generations exceeds N_{gen}^s, S-GMMA terminates.

The kernel function **S-GMMA-LS** is provided in Algorithm 3. For each kernel call, $|P_d|$ threads are created and each thread can map to a unique solution in population P_d according to the thread index. In each thread, the corresponding solution s is optimized $(LS_{iter}/4)$ times using four local search operators (i.e., $LS_1 - LS_4$). Note that the local search procedure in **S-GMMA-LS** is slightly different from that in CMMA. Observe lines 7 – 10 of Algorithm 1, an LS operator is randomly selected and applied on solution s. However, directly using such codes may result in an excess of *thread divergence* [11], because the switch of k will cause many branches to different instructions. The implementation in Algorithm 3 can help to reduce the thread divergence, which is important in optimizing the performance of GPU execution.

S-GMMA invokes N_{gen}^s kernel calls. Each call launches $|P_d|$ $(= |P|)$ threads executed in parallel. And each thread includes LS_iter LS operators. Therefore, LS operators are performed $N_{gen}^s \cdot |P| \cdot LS\_iter$ times in total.

Algorithm 2 The framework of the S-GMMA for solving VRPRB

```
1:  Generate an initial population $P_h$;
2:  while the number of generations does not exceed $N_{gen}^s$ do
3:      $P_h' =$ recombination($P_h$);
4:      $P_d \leftarrow P_h'$; // Use cudaMemcpy to copy data from host to device
5:      Generate a weight $w = (w_1, w_2)$;
6:      S-GMMA-LS <<< $dim3(1), dim3(x_2^s, y_2^s)$ >>> $(P_d, w)$; // $x_2^s \cdot y_2^s \geq |P|$
7:      $P_h' \leftarrow P_d$; // Use cudaMemcpy to copy data from device to host
8:      $P_h =$ update($P_h, P_h'$);
9:  end while
10: return $P_h$;
```

Algorithm 3 The kernel function **S-GMMA-LS**

```
1: $s \leftarrow$ the solution that the thread maps to;
2: for $l = 1$ to $(LS_{iter}/4)$ do
3:     LS$_1(s, w)$;
4:     LS$_2(s, w)$;
5:     LS$_3(s, w)$;
6:     LS$_4(s, w)$;
7: end for
```

5.3 A revised GPU-based multiobjective memetic algorithm

S-GMMA is easy to implement, however, it has some deficiencies. First, it does not fully take advantage of the computing power of GPU. In Algorithm 2, only $|P|$ threads are created, but an ordinary GPU can support more than several thousand threads run in parallel. Second, although we try to reduce the thread divergence in S-GMMA, the combination of local search operators (line 2 – 7 in Algorithm 3) still generates many divergent branches, which may slow down the execution. Third, in CMMA, the tournament selection procedure may repeatedly choose a particular solution for the local refinements. In contrast, S-GMMA puts the same computational efforts on every solution within a generation. Then some elitist solutions may not have enough chances to get further improvements.

To overcome the above deficiencies, we propose a revised version of GMMA (R-GMMA for short). The framework of R-GMMA is shown in Algorithm 4. The loops from line 8 – 15 remove the tournament selection procedure in CMMA. Instead, they perform local improvements on all the solutions in the population P_d. The variable $Iter$ controls the level of computational efforts on local improvements. In early generations, $Iter$ should be large, because every solution could have great potentials to become elitist solutions and heavy computational efforts have to be made. In later generations, $Iter$ can be small, since it is generally difficult to locally improve the solutions and it is not necessary to consume many computational resources. To this end, **func**(g) in line 7 is better set to a decreasing function of g.

In the inner loops (line 10 – 14), we further include a *solution-level parallel model* [29] to parallelize the local search. Observe that an LS operator is applied on at most two random routes of a solution, so we first shuffle array v to randomly pair up routes (line 12), then the LS operators can parallelly perform on $|P|$ solutions, each with $m/2$ pairs of routes (line 13). The kernel function **R-GMMA-LS**$_k$ is given in Algorithm 5 and the associated launch parameters are $dim3(x_1^r)$ and $dim3(x_2^r, m/2)$. Note that k is not a parameter but a fixed value in **R-GMMA-LS**$_k$. This means that the branches of k are made from line 13 of Algorithm 4 but not from line 7 of Algorithm 5. Thus, the thread divergence within GPU can be significantly reduced.

The R-GMMA has N_{gen}^r generations. Each generation has **func**$(g) \cdot LS_{iter}/(m/2)$ kernel calls. Each call creates $|P| \cdot (m/2)$ threads and each thread includes only one LS operator. To sum up, LS operators are performed $\sum_{g=1}^{N_{gen}^r}$ **func**$(g) \cdot |P| \cdot LS_{iter}$ times in total.

Algorithm 4 The framework of the R-GMMA for solving VRPRB

```
Generate an initial population $P_h$;
Set the current number of generations $g = 0$;
Set array $v$ to $(0, \ldots, m-1)$; // array $v$ automatically pair up to $(v_0, v_1), (v_2, v_3), \ldots$
while the number of generations $g$ does not exceed $N_{gen}^r$ do
    $P_h' = $ recombination($P_h$);
    $P_d \leftarrow P_h'$; // Use cudaMemcpy to copy data from host to device
    $Iter \leftarrow$ func($g$);
    for $i = 1$ to $Iter$ do
        Generate a weight $w = (w_1, w_2)$;
        for $l = 1$ to $LS_{iter}/(m/2)$ do
            $k \leftarrow$ generate a random integer in the range $[1, K]$;
            random_shuffle($v$); // Shuffle to randomly pair up routes
            R-GMMA-LS$_k$ $<<<$ $dim3(x_1^r), dim3(x_2^r, m/2)$ $>>>$ $(P_d, v, w)$; //$x_1^r \cdot x_2^r \geq |P|$
        end for
    end for
    $P_h' \leftarrow P_d$; // Use cudaMemcpy to copy data from device to host
    $P_h = $ update($P_h, P_h'$);
    $g \leftarrow g + 1$;
end while
return $P_h$;
```

Algorithm 5 The kernel function **R-GMMA-LS**$_k$

```
$s \leftarrow$ the solution that the thread maps to;
$id \leftarrow$ the index of array $v$ that the thread maps to;
$v_1 \leftarrow v[id * 2]$; // retrieve the index of the first route
$v_2 \leftarrow v[id * 2 + 1]$; // retrieve the index of the second route
$r_1 \leftarrow$ route $v_1$ in solution $s$;
$r_2 \leftarrow$ route $v_2$ in solution $s$;
LS$_k(s, r_1, r_2, w)$;
```

6 Computational experiments

The proposed algorithms were implemented in C++. All the experiments were conducted on a personal computer with an Intel i5-4200 2.80 GHz CPU, 8 GB RAM, and Windows 10 operating system. The GPU model is NVIDIA GeForce GTX 950M and the CUDA version is 8.0.

The test data of VRPRB can be directly obtained from various existing VRP instances. Because [12] conducted experiments on seven VRPRB benchmark instances, we also used these instances to test the performance of our algorithms. All these instances have the same form of names denoted as "Ei-jk", where the character E means that the distance metric is Euclidean, i is the number of customers, j is the number of vehicles and k is an identifier. For each instance, 20 independent runs with different random seeds were made for each algorithm. All the computation time reported in the tables is in seconds.

6.1 Parameter settings

We first discuss the parameter settings of CMMA. Generally speaking, increasing any of the following parameters: the number of generations N_{gen}, the population size $|P|$, the times of tournament selection $Iter$ and the number of local improvements LS_iter, can potentially produce better solutions but bring in additional running time. Similar to the parameters used in [26], we set $|P| = 200$, $LS_{iter} = 3000$, $Iter = 80$ and $N_{gen} = 12000$. Such settings can yield satisfactory solutions within reasonable running time.

The parameter τ $(0 < \tau \leq 1)$ within the tournament selection procedure was tuned as follows. Three levels of τ, i.e., $n/4$, $n/2$ and $3n/4$, were tested. When τ is small, the tournament tends to apply random selection. When τ grows larger, the tournament performs more greedily for choosing elite solutions. Table 1 presents the gaps of two objectives for different τ values. The gap is calculated by averaging the best-found objective with respect to the corresponding τ over the best-known objective on all the instances. The table shows that $\tau = n/2$ can lead to relatively small gaps considering both objectives, so we set τ to $n/2$ in our final settings.

For GMMAs, the parameters which appear in CMMA also follow their settings in CMMA. Those parameters solely used in GMMAs are set as follows: $x_2^s = 32$, $y_2^s = 7$, $x_1^r = 7$, $x_2^r = 32$ $(32 \times 7 \geq 200 = |P|)$, and **func**$(g) = \lceil 60 * 0.9925^g \rceil$.

Note that the settings $x_2^s = x_2^r = 32$ relate to the *warp* size of GPU, which is a sub-division used in the hardware implementation to coalesce memory access and instruction dispatch. Such settings can better utilize the parallel capability of GPU. **func**(g) is set to a geometric decreasing function. The settings of N_{gen}^s and N_{gen}^r are discussed in Section 6.3.

Table 1 Parameter tuning for different τ values

	$\tau = n/4$	$\tau = n/2$	$\tau = 3n/4$
First objective gap	0.17%	0.08%	0.03%
Second objective gap	0.07%	0.05%	0.14%

Table 2 Performance comparison between Jozefowiez et al. [12]'s method and CMMA on VRPRB instances (Minimization of the total traveling distance)

Instance	Solely minimizing the total traveling distance				
	Best known	Results from Jozefowiez et al. [12]		CMMA	
		Best found	Associated difference	Best found	Associated difference
E51-05e	524.61	524.61	20.07	524.61	20.07
E76-10e	835.26	835.32	78.10	835.32	78.10
E101-08e	826.14	827.39	67.55	**826.14**	97.89
E101-10c	819.56	819.56	93.43	819.56	93.43
E121-07c	1042.11	1042.11	146.67	1042.11	146.67
E151-12c	1028.42	1047.35	74.78	**1042.32**	82.10
E200-17c	1291.45	1352.46	76.60	**1324.48**	75.64

6.2 Computational results of VRPRB

The first set of experiments is to test the quality of the solutions found by CMMA. We compare our results with the results given by the referenced approach Jozefowiez et al. [12]. In their problem setting, the balancing objective is set to the difference between the maximal and the minimal route cost (see Expression (12)), which is different from our description of VRPRB. Nevertheless, we modified the second objective to Expression (12) and executed our programs.

Since VRPRB has two objectives, we only report two extreme solutions for each instance, i.e., solely minimizing the total traveling cost and solely minimizing the difference between the maximal and the minimal route cost. The results are shown in Tables 2 and 3.

In Table 2, the column "Best known" gives the currently best total traveling cost found by the existing VRP approaches. The column "Best found" indicates the best value of the first objective found by the algorithm, and the associated second objective value is shown in the column "Associated difference". Similarly, the column "Best found" in Table 3 presents the best value of the second objective and the column "Associated traveling cost" is the associated first objective.

In both tables, the numbers are marked in bold if the best found value is better than that of the other approach. From the tables, we can find that our CMMA approach outperforms Jozefowiez et al. [12]'s method on all the instances in terms of the quality of the two extreme solutions.

The problem setting by Jozefowiez et al. [12] may lead to distorted solutions. We depict in Fig. 5a and b the two extreme solutions produced by our approach on instance E76-10e, respectively. We can observe from Fig. 5b that when the second objective is defined by Expression (12), many routes take detors for achieving route balancing, then a *distorted* solution is produced.

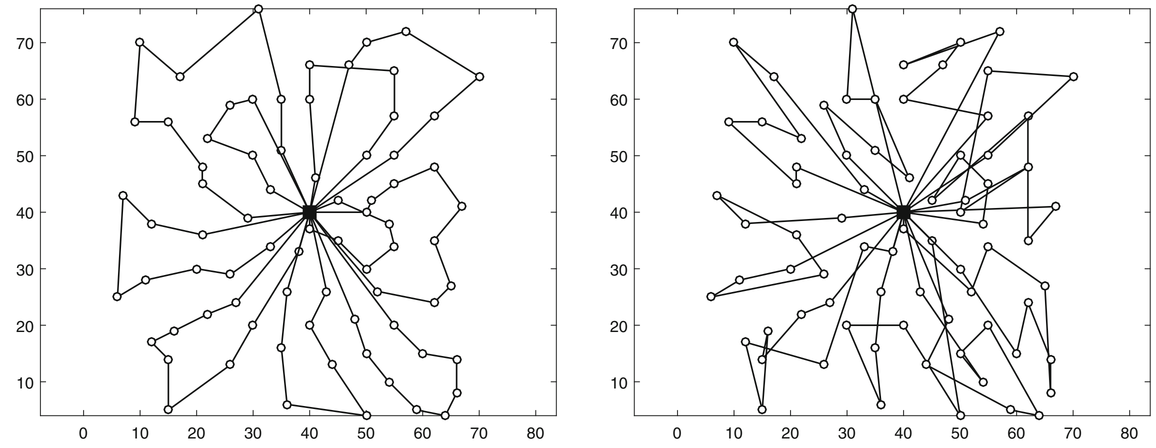
We therefore changed the balancing objective back to Expression (11) and re-executed our algorithm. The two extreme solutions with respect to each instance are reported in Table 4. We also take instance E76-10e as an example, the two extreme solutions found are (835.32, 119.32) and (865.62, 94.08), which are detailedly illustrated in Fig. 6a and b. These figures demonstrate that by setting the balancing objective to Expression (11), the occurrence of detors can be effectively avoided.

6.3 Comparisons among CMMA and GMMAs

The second set of experiment is to compare CMMA with two GMMAs. Two performance indicators, called inverted generational distance (*IGD*) and hypervolume (*HV*), are adopted

Table 3 Performance comparison between Jozefowiez et al. [12]'s method and CMMA on VRPRB instances (Minimization of the difference between the maximal and minimal route cost)

Instance	Solely minimizing the difference			
	Results from Jozefowiez et al. [12]		CMMA	
	Best found	Associated traveling cost	Best found	Associated traveling cost
E51-05e	0.24	618.22	**0.013**	791.05
E76-10e	0.59	1203.98	**0.075**	1142.93
E101-08e	0.29	1871.06	**0.042**	1297.47
E101-10c	1.15	1429.90	**0.029**	1378.36
E121-07c	0.10	2388.30	**0.004**	1598.49
E151-12c	0.80	1484.48	**0.040**	1526.14
E200-17c	1.38	1902.64	**0.070**	1889.03



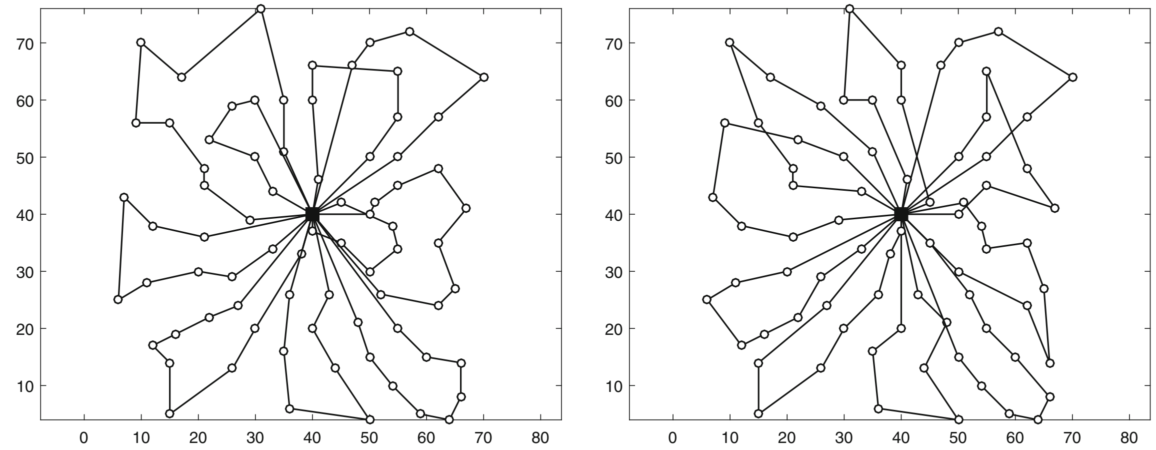
(a)

(b)

Fig. 5 **a** The solution for instance E76-10e with the minimum traveling cost. **b** The solution for instance E76-10e with the minimum difference

Table 4 The computational results on VRPRB instances

Instance	Solely minimizing the total traveling cost		Solely minimizing the maximal route cost	
	Best found	Associated maximal route cost	Best found	Associated traveling cost
E51-05e	524.61	118.52	111.37	537.89
E76-10e	835.32	119.32	94.08	865.62
E101-08e	826.14	138.79	113.33	869.16
E101-10c	819.56	137.02	120.72	942.96
E121-07c	1042.12	213.63	200.84	1243.65
E151-12c	1031.96	120.62	102.82	1092.72
E200-17c	1312.88	119.41	99.86	1496.10



(a)

(b)

Fig. 6 **a** The solution for instance E76-10e with the minimum traveling cost. **b** The solution for instance E76-10e with the minimum of maximal route cost

Table 5 The comparison of *IGD* among different algorithms

Instance	*IGD*				
	CMMA	S-GMMA (2400)	S-GMMA (6000)	R-GMMA (120)	R-GMMA (6000)
E51-05e	0.3139	0.0870	**0.0502**	0.3240	0.2770
E76-10e	0.1489	0.1681	**0.1305**	0.4155	0.2380
E101-08e	0.0899	0.1466	0.1416	0.1137	**0.0753**
E101-10c	**0.0464**	0.1238	0.1173	0.1508	0.0485
E121-07c	0.1604	0.1845	0.1410	0.2413	**0.1073**
E151-12c	0.1310	0.2675	0.2233	0.1575	**0.0711**
E200-17c	**0.1113**	0.3514	0.2860	0.1788	0.1196
Average	0.1431	0.1898	0.1557	0.2259	**0.1338**

to numerically evaluate the performance of different algorithms. These indicators are briefly introduced as follows.

1. *Inverted generational distance (IGD)* [5]. The IGD indicator estimates the distance of the elements in an approximated Pareto front toward those in the true Pareto front. A smaller *IGD* indicates a better approximated Pareto set.
2. *Hypervolume (HV)* [32]. For bi-objective minimization problems, the *HV* value is given by the area of the union of all rectangles covered by the Pareto optimal solutions. Both convergence and diversity of the non-dominated solutions can be reflected from the *HV* value. The larger the *HV* value is, the closer the corresponding non-dominated solution set is to the Pareto front.

According to the above discussions, the true Pareto front is required for calculating *IGD* and *HV*. However, the true Pareto front is unknown. Hence, for each instance, the non-dominated solutions produced by all the algorithms over 20 runs were collected to form the front.

We hereby provide five algorithms for the comparison, which are CMMA, S-GMMA (2400), S-GMMA (6000), R-GMMA (120) and R-GMMA (6000). Note that the number inside the bracket denotes the number of generations for the corresponding algorithm. The rationales behind such generation settings are as follows. For S-GMMA (2400) and R-GMMA (120), the corresponding number of LS operators invoked is the same as that for CMMA. It is equivalent to solving $N_{gen} \cdot Iter \cdot LS_{iter} = N_{gen}^{s} \cdot |P| \cdot LS\_iter = \sum_{g=1}^{N_{gen}^{r}} \mathbf{func}(g) \cdot |P| \cdot LS_{iter}$. Because $N_{gen} = 12000$ for CMMA, we have $N_{gen}^{s} = 2400$ and $N_{gen}^{r} = 120$. For S-GMMA (6000) and R-GMMA (6000), the corresponding execution time is roughly the same as that for CMMA.

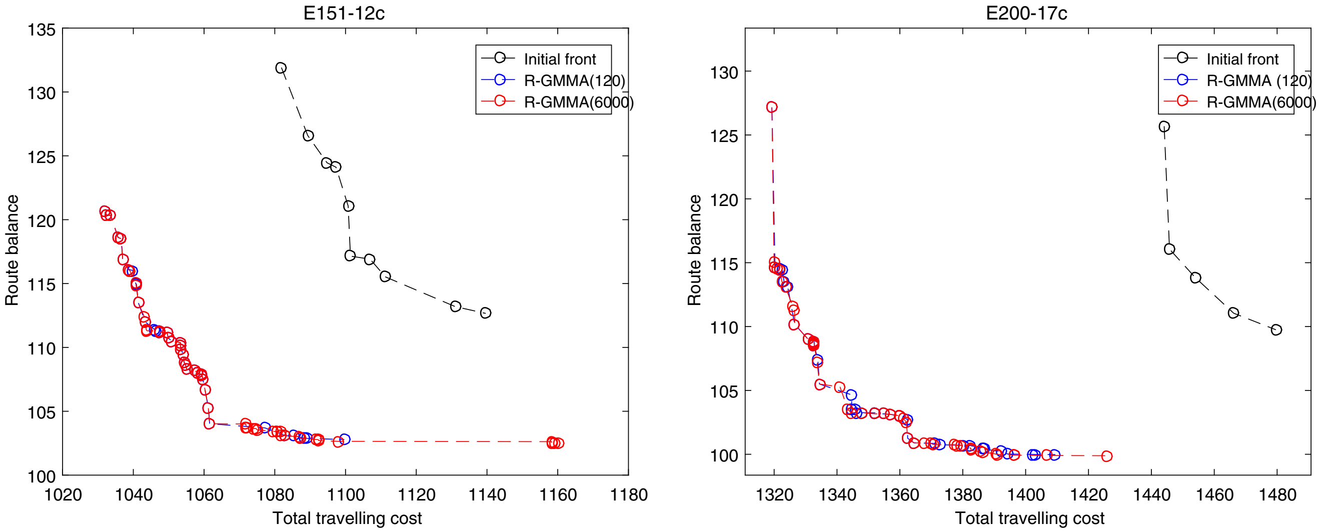
Tables 5, 6 and 7 respectively show the values of *IGD*, *HV* and average execution time of all the algorithms. In these tables, the smallest *IGD* and the largest *HV* for each instance are marked in bold. We can find that for those algorithms with the same number of LS operators, CMMA performs better than S-GMMA (2400) and R-GMMA (120). This is because the serial implementation can take advantage of the tournament selection procedure but the parallel implementation cannot. However, the average computation time of S-GMMA (2400) and R-GMMA (120) are much faster than those of CMMA. Next, let us consider CMMA, S-GMMA (6000) and R-GMMA (6000). Their average execution time is around 750. R-GMMA (6000) performs better than S-GMMA (6000) and CMMA in terms of the average

Table 6 The comparison of *HV* among different algorithms

Instance	*HV*				
	CMMA	S-GMMA (2400)	S-GMMA (6000)	R-GMMA (120)	R-GMMA (6000)
E51-05e	0.1968	0.4584	**0.5351**	0.2108	0.2108
E76-10e	0.6864	0.6849	0.7442	0.3429	**0.7519**
E101-08e	0.6374	0.5652	0.5740	0.6751	**0.6989**
E101-10c	0.8612	0.8723	**0.8767**	0.7436	0.8603
E121-07c	0.3009	0.2698	0.3140	0.2044	**0.3520**
E151-12c	0.5314	0.3732	0.4189	0.5410	**0.6226**
E200-17c	**0.8062**	0.5633	0.6442	0.6588	0.8028
Average	0.5743	0.5410	0.5867	0.4824	**0.6142**

Table 7 The comparison of average execution time among different algorithms

Instance	Time				
	CMMA	S-GMMA (2400)	S-GMMA (6000)	R-GMMA (120)	R-GMMA (6000)
E51-05e	617.88	234.99	595.04	485.75	1103.23
E76-10e	657.64	265.88	686.25	218.51	565.85
E101-08e	724.45	289.19	714.54	285.04	702.86
E101-10c	751.40	297.12	728.35	235.58	610.49
E121-07c	756.20	329.03	805.24	369.44	909.81
E151-12c	869.86	342.63	808.82	217.83	633.80
E200-17c	1051.93	396.22	936.67	178.42	634.50
Average	775.62	307.87	753.56	284.37	737.22

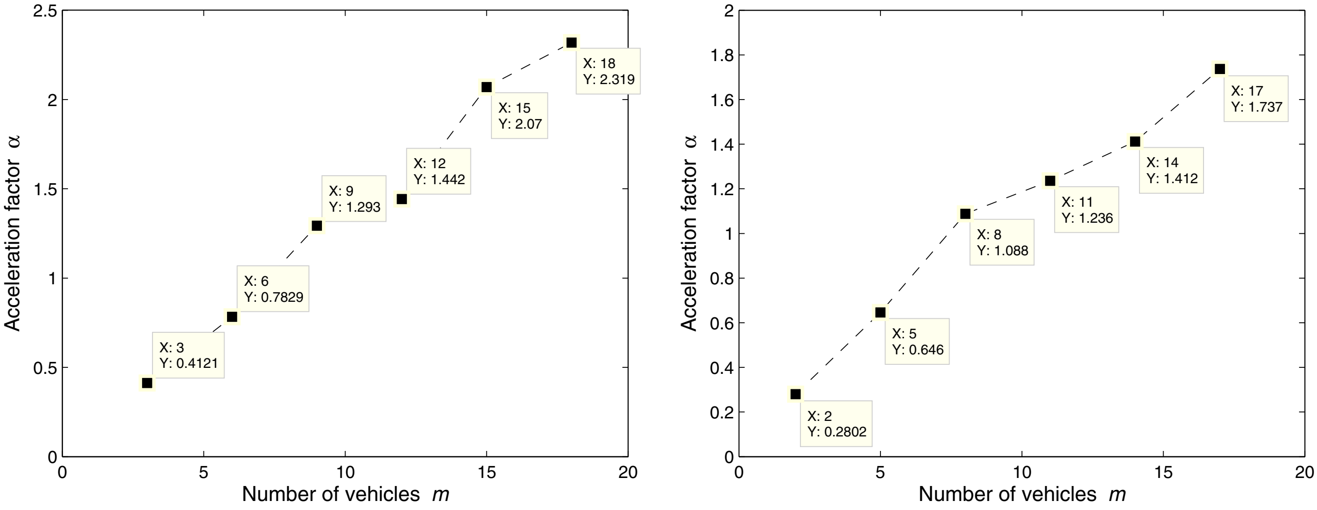


(a) E151-12c (b) E200-17c

Fig. 7 Approximated Pareto front obtained by different algorithms

Table 8 The execution time of CMMA and R-GMMA and the acceleration rate on six instances

Instance	Time of CMMA	Time of R-GMMA(6000)	α
E151-3c	922.63	2238.71	0.4121
E151-6c	924.25	1180.55	0.7829
E151-9c	919.93	711.33	1.2932
E151-12c	924.66	641.04	1.4424
E151-15c	932.44	450.46	2.0700
E151-18c	937.01	404.09	2.3188
E200-2c	1058.85	3778.34	0.2802
E200-5c	1108.33	1715.80	0.6460
E200-8c	1104.59	1015.23	1.0880
E200-11c	1098.96	889.39	1.2356
E200-14c	1087.66	770.30	1.4120
E200-17c	1101.99	634.50	1.7368



(a) E151 (b) E200

Fig. 8 The trend of the acceleration rate along with the growth of the number of vehicles

IGD or *HV* value. In addition, when the scale of problems enlarges, the performance of R-GMMA becomes much better than S-GMMA. This can be attributed to the positive effect of the controlled function **func(g)**.

We further pictorialize the approximated Pareto fronts obtained by R-GMMA (120) and R-GMMA (6000) on instances E151-12c and E200-17c, as shown in Fig. 7. The front of the initial population is also depicted. It can be seen that both R-GMMA (120) and R-GMMA (6000) greatly push the initial front toward left and bottom directions, which means that two objectives get significantly improved. We also notice that the *extent* of the approximated Pareto front by R-GMMA (6000) is greater than that by R-GMMA (120). This indicates that R-GMMA (6000) can produce a set of non-dominated solutions with better diversity.

6.4 Analysis of the GPU acceleration

We would like to analyze the acceleration rate of R-GMMA (6000) to CMMA when the structure of an instance varies. The acceleration rate α is defined by:

$$\alpha = \frac{t_{CMMA}}{t_{R-GMMA}} \tag{17}$$

where t_{CMMA} and t_{R-GMMA} represent the execution time of CMMA and R-GMMA, respectively.

We selected two largest scale instance, i.e., E151-12c and E200-17c for the analysis. E151-12c has $n = 151$ customers, $m = 12$ vehicles and the vehicle capacity is $C = 200$. We altered the number of vehicles and the vehicle capacity but remained other information unchanged, thereby constructing six new instances (we named them E151-2c, E151-5c, E151-8c, E151-11c, E151-14c and E151-17c). The construction is as follows. When the number of vehicles changed from m to m', the new vehicle capacity was set to $\lceil \frac{Cm}{m'} \rceil$. For E200-17c, similar manner was adopted to create six instances.

Table 8 shows the values of t_{CMMA}, t_{R-GMMA} and α on these instances. Figure 8 further illustrates the trend of the acceleration rate as the number of vehicles increases. In R-GMMA, $m/2$ pairs of routes are optimized in parallel. When m is small, R-GMMA ran much slower than CMMA did. The acceleration rate increases steadily along with the growth of the number of vehicles. When m grows large, high acceleration rate is exhibited, which shows the power of parallel computing.

7 Conclusions

In this paper, we analyze the structure of a vehicle routing problem with route balancing (VRPRB). Some interesting finding about the distorted solution is discussed. We propose a multiobjective memetic algorithm (MMA) and try to parallelize it based on GPU computing. The experiments support our finding and demonstrate that GPU computing is a good option to parallelize the algorithm. Our methodology can certainly be extended to solve other kinds of complex multiobjective optimization problems.

In practice, programming with GPU should consider not only the domain knowledge of the problem, but also the architecture of GPU devices. Our research is a preliminary attempt to the GPU computing. Much more efforts can be made to further accelerate the execution of algorithms. For example, adaptive methods can be employed to dynamically make full use of GPU resources.

Acknowledgements An earlier version of this paper has been presented at IEA/AIE 2017. The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported by the National Science Foundation of China (No. 71601191, 61673403), Natural Science Foundation of Guangdong Province (No. 2016A030313264), the Opening Project of Guangdong High Performance Computing Society (No. 2017060109) and Guangzhou Science and Technology Project (No. 2016201604030034).

References

1. Beasley JE (1983) Route firstcluster second methods for vehicle routing. Omega 11(4):403–408
2. Benaini A, Berrajaa A, Daoudi EM (2016) Solving the vehicle routing problem on gpu. In: Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015. Springer, pp 239–248
3. Bräysy O, Gendreau M (2005) Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transplant Sci 39(1):104–118
4. Brodtkorb AR, Hagen TR, Schulz C, Hasle G (2013) Gpu computing in discrete optimization. part i: Introduction to the gpu. EURO J Trans Log 2(1-2):129–157
5. Coello CAC, Cortés NC (2005) Solving multiobjective optimization problems using an artificial immune system. Genet Program Evolvable Mach 6(2):163–190
6. Deb K (2001) Multi-objective optimization using evolutionary algorithms, vol 16. Wiley, New York
7. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
8. Ermiš G, Ċatay B (2017) Accelerating local search algorithms for the travelling salesman problem through the effective use of gpu. Trans Res Procedia 22:409–418
9. Garcia-Najera A, Bullinaria JA (2009) Bi-objective optimization for the vehicle routing problem with time windows: Using route similarity to enhance performance. In: International Conference on Evolutionary Multi-Criterion Optimization. Springer, pp 275–289
10. Guide D (2017) Cuda c programming guide. NVIDIA June
11. Jason S, Edward K (2010) Cuda by example: an introduction to general-purpose gpu programming
12. Jozefowiez N, Semet F, Talbi EG (2009) An evolutionary algorithm for the vehicle routing problem with route balancing. Eur J Oper Res 195(3):761–769
13. Kritikos MN, Ioannou G (2010) The balanced cargo vehicle routing problem with time windows. Int J Prod Econ 123(1):42–51
14. Lacomme P, Prins C, Prodhon C, Ren L (2015) A multi-start split based path relinking (msspr) approach for the vehicle routing problem with route balancing. Eng Appl Artif Intel 38:237–251
15. Lee TR, Ueng JH (1999) A study of vehicle routing problems with load-balancing. Int J Phys Distrib Logist Manag 29(10):646–657
16. Mandal SK, Pacciarelli D, Lkketangen A, Hasle G (2015) A memetic nsga-ii for the bi-objective mixed capacitated general routing problem. J Heuristics 21(3):359–390
17. Melián-Batista B, De Santiago A, AngelBello F, Alvarez A (2014) A bi-objective vehicle routing problem with time windows: A real case in tenerife. Appl Soft Comput 17:140–152
18. Miller BL, Goldberg DE et al (1995) Genetic algorithms, tournament selection, and the effects of noise. Complex Syst 9(3):193–212
19. Miller CE, Tucker AW, Zemlin RA (1960) Integer programming formulation of traveling salesman problems. J Acm 7(4):326–329
20. Moscato P et al (1989) On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concur Comput Program, C3P Rep 826:1989
21. Oyola J, Løkketangen A (2014) Grasp-asp: An algorithm for the cvrp with route balancing. J. Heuristics 20(4):361–382
22. Pacheco J, Caballero R, Laguna M, Molina J (2013) Bi-objective bus routing: an application to school buses in rural areas. Transp Sci 47(3):397–411
23. Rios E, Ochi LS, Boeres C, Coelho VN, Coelho IM, Farias R (2018) Exploring parallel multi-gpu local search strategies in a metaheuristic framework. J Parallel Distrib Comput 111:39–55
24. Schulz C (2013) Efficient local search on the gpuinvestigations on the vehicle routing problem. J Parallel Distrib Comput 73(1):14–31
25. Schulz C, Hasle G, Brodtkorb AR, Hagen TR (2013) Gpu computing in discrete optimization. part ii: Survey focused on routing problems. EURO J Transp Logist 2(1-2):159–186
26. Sun Y, Liang Y, Zhang Z, Wang J (2017) M-nsga-ii: A memetic algorithm for vehicle routing problem with route balancing. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer, pp 61–71
27. Szymon J, Dominik Ż (2013) Solving multi-criteria vehicle routing problem by parallel tabu search on gpu. Procedia Comput Sci 18:2529–2532
28. Toth P, Vigo D (2002) The Vehicle Routing Problem. Discrete mathematics and applications. Society for Industrial and Applied Mathematics. https://books.google.com/books?id=ZzOGQgAACAAJ
29. Van Luong T, Melab N, Talbi EG (2013) Gpu computing for parallel local search metaheuristic algorithms. IEEE Trans Comput 62(1):173–185
30. Wodecki M, Bożejko W, Karpiński M, Pacut MWyrzykowski R, Dongarra J, Karczewski K, Waśniewski J (eds) (2014) Multi-gpu parallel memetic algorithm for capacitated vehicle routing problem. Springer Berlin Heidelberg, Berlin
31. Zhou W, Song T, He F, Liu X (2013) Multiobjective vehicle routing problem with route balance based on genetic algorithm Discrete Dynamics in Nature and Society
32. Zitzler E, Laumanns M, Thiele L et al (2001) Spea2: Improving the strength pareto evolutionary algorithm. In: Eurogen, vol 3242, pp 95–100

Zizhen Zhang received B.S. and M.S. degree from Sun Yat-sen University, China, in 2007 and 2009, respectively. He received a Ph.D degree from City University of Hong Kong in 2014. He is currently an Associate Professor in the School of Data Science and Computer Science, Sun Yat-sen University, China. His research interests include computational intelligence and its applications in production, transportation and logistics.

Yuyan Sun received the B.S. degree in the School of Data and Computer Science, Sun Yat-sen University, in 2017. She is currently pursuing the M.S. degree. Her research interests include multiobjective optimization and its application to vehicle routing problems.

Hong Xie is purchasing the B.S. degree in the School of Data and Computer Science from Sun Yat-sen University. His research interest is computational intelligence.

Yi Teng received B.S. in the Department of Computing Science from Xi Dian University in 2007, and M.S. degree in the Department of Computer Science from Sun Yat-sen University in 2009. She received a Ph.D degree from The Hong Kong Polytechnic University in 2015. She is currently a senior engineer in China Post Group Co., Ltd. Her research interests include data analysis and data mining.

Jiahai Wang received the Ph.D. degree from Toyama University, Toyama, Japan, in 2005. In 2005, he joined Sun Yat-Sen University, Guangzhou, China, where he is currently a Professor. His main research interests include computational intelligence and its applications.