Solving the Uncapacitated Single Allocation p-Hub Median Problem on GPU

Chapter *in* Studies in Computational Intelligence · January 2019

DOI: 10.1007/978-3-319-95104-1_2

CITATIONS
0

READS
80

4 authors:

Benaini Abdelhamid
Normandie Université, le havre
33 PUBLICATIONS **137** CITATIONS
SEE PROFILE

Achraf Berrajaa
Université Mohammed Premier / Université du Havre
9 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Jaouad Boukachour
Université du Havre
112 PUBLICATIONS **336** CITATIONS
SEE PROFILE

Mustapha Oudani
Université Internationale de Rabat
32 PUBLICATIONS **81** CITATIONS
SEE PROFILE

Chapter 2
Solving the Uncapacitated Single Allocation p-Hub Median Problem on GPU

A. Benaini, A. Berrajaa, J. Boukachour and M. Oudani

Abstract This chapter presents a parallel GPU implementation for solving the Uncapacitated Single Allocation p-Hub Median Problem with genetic algorithm. Starting from an initial solution that locates hubs at center nodes, our GPU implementation quickly reaches optimal or near-optimal solutions. It efficiently exploits the computing power of the GPU and outperforms, in cost and in computing time, the other approaches proposed in the literature for this problem. We compare it to the best recent solutions on the benchmarks up to 1000 nodes and on large instances up to 6000 nodes generated by us. Moreover, we solved instance problems so far unsolved.

2.1 Introduction

Hubs are sort of facilities that serve to transfer transshipment and sort in many-to-many complex distribution networks. They find applications in airline passengers, telecommunications, distributed computing and postal delivery networks. In air traffic, hubs are the central airports for long haul by cargo planes. In the telecommunication and distributed computing networks, hubs may be concentrators, servers, routers or multiplexers, etc. In postal distribution networks, hubs are sorting and the cross dock centers. The hub location problem can be stated as follows. Given a set of N nodes and the corresponding cost flows w_{ij} between nodes i and j, $1 \leq i, j \leq N$,

A. Benaini (✉) · J. Boukachour
Normandie University, LMAH, Le Havre, France
e-mail: abdelhamid.benaini@univ-lehavre.fr

J. Boukachour
e-mail: jaouad.boukachour@univ-lehavre.fr

A. Berrajaa
Normandie University and University Mohammed 1er, LaRI, Oujda, Morocco
e-mail: berrajaa.achraf@gmail.com

M. Oudani
Private University of Marrakech, Marrakech, Morocco
e-mail: oudani.mustapha@gmail.com

E.-G. Talbi and A. Nakib (eds.), *Bioinspired Heuristics for Optimization*,
Studies in Computational Intelligence 774,
https://doi.org/10.1007/978-3-319-95104-1_2

the objective is to locate p appropriate nodes in a network to be hubs and to allocate the non hub nodes to hubs such that the total flow-weighted costs across the network is minimized. The success of this type of network is due to the economy of scale achieved by consolidating the traffic through hub-hub arcs. Indeed, in this network any flow between pair of nodes can only take place through the hubs. The cost from an origin non-hub i to a destination non-hub j through two hubs k and l is expressed as $C_{ijkl} = \chi w_{ik} + \alpha w_{kl} + \gamma w_{lj}$ where $\alpha < 1$ represents the scale economy generated by consolidating flows between hubs, while χ and γ represent the distribution and the collection costs respectively and are often greater than 1.

Several studies concerning hub location problems have been developed since 1980. Different variants of hub location problems have been defined and classified according to the method of allocation into two main categories. The single allocation, where each non hub is assigned to exactly one hub, while the multiple variant enables the non-hubs to be allocated to several hubs. The problem is said to be uncapacitated if the hubs have infinite capacities; otherwise it is capacitated. Moreover, if the number of hubs p is given then the problem is called Uncapacitated Single Allocation p-Hub Median Problem (USApHMP); otherwise the problem is said to be Uncapacitated Single Allocation Hub Location Problem (USAHLP). In this last case, the number of hubs and their locations are decision variables that must be defined. This is formulated by adding an additional term in the objective function, aiming to minimize the fixed cost of installing hubs.

Other versions of hub problems have been introduced in the literature such as (1) p-hub center problem [1] where the objective is to minimize the maximum travel time between two demand centers (2) The hub arc problem [2] which aims to overtake the shortcomings of the p-hub median problem by introducing the bridges arcs between hubs without any discount factor (3) The dynamic hub location problem [3] where either cost, demands or resources may vary in the planning horizon (4) With congestion, where economic penalties are imposed if hubs with congestion levels are used; (5) With non-linear costs with stochastic elements; (6) With vehicle routing constraints [4]. Reviews, synthesis and methods for solving these problems are presented earlier [5].

As the USApHMP is NP-hard, it cannot be solved to optimality (by exact methods) for realistically sized instances in reasonable time. Exact methods provide optimal solutions for smaller size problem instances with $N \leq 200$ nodes. So, many of the solution techniques suggested for this problem are heuristics. Among the most promising and successful heuristics are Tabu Search, Simulated Annealing, Genetic Algorithm and Greedy methods. This chapter will focus on USApHMP for which we propose a parallel genetic algorithm and it GPU implementation. To our knowledge, this is the first GPU implementation that solves this problem. We show under tests on known benchmarks the efficiency of our approach. Note that the GPUs and the underlying parallel programming model Cuda are actually available in most personal computers. So, massively parallel computing has become a commodity technology.

The remainder of this chapter is organized as follows. Related works are provided in Sect. 2.2. In Sect. 2.3, we present the mathematical formulation of the problem. The GA description is presented in Sect. 2.4, followed by the GPU implementation

in Sect. 2.5. Computational results are reported in Sect. 2.6 and Concluding remarks are given in Sect. 2.7.

2.2 Related Works

The first mathematical formulation for the USApHMP as a quadratic integer program was presented in [6]. Two heuristics were developed to solve it with numerical results for CAB (Civilian Aeronautics Board) data. Reference [1] presented integer formulations for the p-hub median problem (pHMP), p-hub center problem and hub covering problems. Possible extensions with flow thresholds were also studied. Also [2] introduced the p-HMP with two heuristics for solving it tested on a dataset with 10-40 nodes and up to 8 hubs. Reference [7] studied the special case of the single allocation two-hub location problem. In this particular case, the quadratic program is transformed to a linear program and to a minimum cut problem. Reference [8] proposed an hybrid genetic algorithm and tabu search heuristic and reported the results on the CAB dataset. Reference [9] presented a solving approach for the multiple allocation p-HMP version and described how this could be adapted to the single allocation case. Their method is based on the shortest-path algorithm to find lower bounds for a branch-and-bound scheme that find the exact solution. The results are reported on AP data for a multiple allocation case up to 200 nodes. Reference [10] studied four models: the first was concerned with a capacitated network, the second focus on the minimum threshold model, the third determined the numbers of open hubs and the last one introduced a flow-dependent cost function.

Reference [11] proposed a model implemented in a GIS environment to prove that hub networks may emerge naturally on traffic networks to take advantage of economies of scale. Reference [12] studied the polyhedral properties of the single allocation hub location problem and proposed a Branch-and-Cut algorithm for solving this problem. Reference [13] proposed a hybrid heuristic to solve the USAHLP. Their method is based on a combination of an upper bound method search, simulated annealing and tabu list heuristic and was tested on CAB data and AP data up to 200 nodes. Reference [14] proposed three variants of tabu search heuristics and a two-stage integrated tabu search to solve the USAHLP. These authors used the multi-start principle to generate different initial solutions that are improved by the tabu search. They reported the results for new introduced larger instances with 300 and 400 nodes. Reference [15] proposed a general variable neighborhood search for the USApHMP. They reported the results on large AP data, PlanetLab instances and Urand instances up to 1000 nodes. Reference [4] introduced the single allocation hub location problem under congestion with a generalized Benders decomposition algorithm to solve AP instances.

Reference [16] proposed a memetic algorithm to solve the USAHLP. Their evolutionary approach is based on two local search heuristics. They tested their algorithm on the well-known benchmark and generated larger scale instances up to 900 nodes. They gave the optimal solutions of AP data up to 200 nodes. Reference

[17] proposed Discrete Particle Swarm Optimization to solve the USAHLP. They obtained the optimal solutions on all CAB datasets and on AP data up to 200 nodes. Reference [18] introduced a planar version of the USAHLP with the particularity that a hub could be located anywhere in the plan. Reference [19] proposed a threshold accepting algorithm to solve the USAHLP and reported the results on the AP and CAB benchmarks.

Reference [20] made use of dataset structures to propose a new linearization of the quadratic formulation of the problem. They obtained optimal solutions on the AP data up to 200 nodes. Reference [21] introduced a new version of USApHMP, where the discount factor between hubs is replaced by a decision variable. They proposed a branch-and-bound algorithm with Lagrangian relaxation to solve the problem. Recently [22] proposed a Tabu Search heuristic for solving the USAHLP and reported the results both on CAB data and an AP dataset up to 400 nodes. Reference [23] presented a greedy deterministic and randomized algorithms for constructing an initial feasible solution and improves it by GRASP heuristic.

References [24, 25] proposed an efficient GA for solving the uncapacited single and multiple allocation hub problems. Binary encoding and adapted genetic operators to these problems are used (only allocation hubs are given as the solution). They showed, under experimental results on ORLIB instances with up to 200 nodes, that the GA approach quickly reaches all known optimal solutions. In this study, we use similar integer encoding and genetic operators to conceive a parallel GA that solves efficiently the USApHMP.

Several studies suggest different strategies to implement GAs on different parallel machines [26–29]. It appears that are three major types of parallel GAs: (1) the master-slave model, (2) the island model and (3) the fine-grained model. In the master-slave model, the master node holds the population and performs most of the GA operations. The fitness evaluation, crossover, correction and mutation operations on groups of individuals are made by each slave. In a coarse-grained model, the population is divided into several nodes. Each node then has a subpopulation on which it executes GA operations. In fine-grained models, each node only has a single individual and can only communicate with several neighboring nodes. In this case, the population is the collection of all the individuals in each node. There are conflicting reports over whether multiple independent runs of GAs with small populations can reach solutions of higher quality, or can find acceptable solutions faster, than a single run with a large population. We adopt the fine-grained model, which is more suitable for the GPU, to conceive the GA for the USApHMP and we solve the USAHLP by solving in parallel all the USApHMP for $p = 1, \ldots, N$. The solution of the USAHLP is the one of the USApHMP with the minimum flow cost among these N solutions.

2.3 The Problem Formulation

The USAHLP can be stated as follows. Given N nodes $1, \ldots, N$. Locate p hubs and allocate each non-hub to a single hub in order to minimize the total flow cost. There

are three levels of decisions in the USAHLP : (i) the number p of hubs to be located; (ii) which nodes will be hubs (iii) how the non-hub nodes will be allocated to hubs. In the USApHMP, the decision concerns only (ii) and (iii), since the hub number p is given.

The USAHLP is formulated as a MIP by Ernst et al. [30] as follows. Let w_{ij} the flow originated from i to j, d_{ij} the distance between i and j and F_k the cost for opening a hub at the node k. Let z_{ik} the binary variable that is equal to 1 if node i is assigned to the hub k and 0 otherwise (with $z_{kk} = 1$ for each hub k). y_{kl}^i the total flow emanating from i that is routed between hubs k and l. $O_i = \sum_j w_{ij}$ the flow departing from origin i and $D_i = \sum_j w_{ji}$ the total cost destined to i.

$$minimize \quad \sum_i \sum_k d_{ik} z_{ik} (\chi O_i + \gamma D_i) + \alpha \sum_i \sum_k \sum_l d_{kl} y_{kl}^i + \sum_k F_k z_{kk} \tag{2.1}$$

Subject to:

$$\sum_k z_{ik} = 1, \ 1 \leq i \leq N \tag{2.2}$$

$$z_{ik} \leq z_{kk}, \ 1 \leq i, k \leq N \tag{2.3}$$

$$\sum_l y_i^{kl} - \sum_l y_i^{lk} = O_i z_{ik} - \sum_j w_{ij} z_{jk}, \ 1 \leq i, k \leq N \tag{2.4}$$

$$z_{ik} \in \{0, 1\}, \ 1 \leq i, k \leq N \tag{2.5}$$

$$y_i^{kl} \geq 0, \ \ 1 \leq i, k, l \leq N \tag{2.6}$$

The objective function (1) minimizes the total cost of flow transportation between all origin- destination nodes plus the cost of establishing the hubs. Constraint (2) imposes each non-hub to be allocated to exactly one hub. Constraint (3) enforces that flow is only sent via open hubs, preventing direct transmission between non-hub nodes. Constraint (4) is the flow conservation constraint and finally constraint (5) and (6) reflect non-negative and binary nature of variables y_{kl}^i and z_{ik} respectively.

The USApHMP (the number of hubs p is given) is formulated as follows:

$$minimize \quad \sum_i \sum_k d_{ik} z_{ik} (\chi O_i + \gamma D_i) + \alpha \sum_i \sum_k \sum_l d_{kl} y_{kl}^i \tag{2.7}$$

subject to

$$\sum_k z_{kk} = p \tag{2.8}$$

and to constraints (2)–(6). Unfortunately these problems are NP-Hard and hence difficult to solve for optimally. So, heuristic and/or parallel approaches are needed to solve large size intances. Figure 2.1a shows a simple example of the USApHMP with $N = 5$ nodes presented by their distances (numbers on the arcs). The number

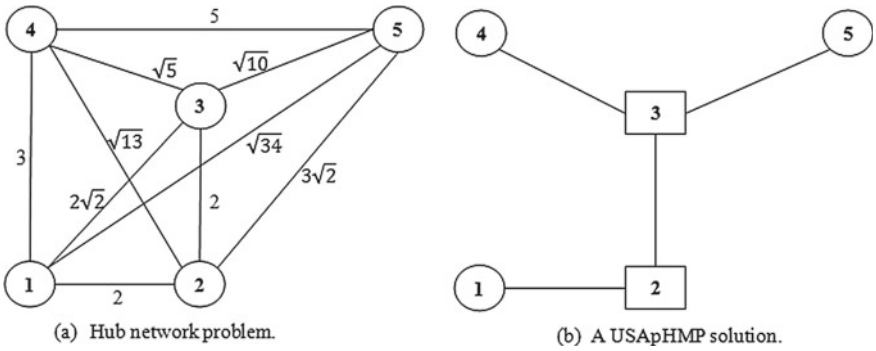

Fig. 2.1 Example of USApHMP solution given in [26]

of hubs that will be located is $p = 2$ and the parameters $\chi = \gamma = 1$ and $\alpha = 0.25$. The amount of flow w_{ij} from a node i to a node j is equal to 1, even for $i = j$. So $O_i = D_i = 5$ for all i. Figure 2.1b presents a solution for this network. The hubs are nodes 2 and 3. The nodes 1 and 2 are allocated to hub 2, while nodes 3, 4 and 5 are allocated to hub 3 (each hub is allocated to itself).The overall transportation cost of this solution is 79.983.

2.4 The Genetic Algorithm

In the following, we outline the GA for solving USApHMP and we show how to use it to solve USAHLP.

Encoding:
Since an encoding of a solution of the USApHMP must express the locations/ allocations, we use a simple encoding by an N-array also used among others in [24, 25, 31]. That is, the encoding S is defined by $S[i] = k$ if node i is allocated to the hub k and $S[k] = k$ if k is hub (each hub is allocated to itself). For instance, the corresponding encoding of the solution of Fig.2.1b is $S = (2, 2, 3, 3, 3)$.

Initial solutions for the parallel GA:
In order to quickly reach an optimal or near-optimal solution, we generate a feasible initial solution with p hubs, say P_p, for the GA as follow.

- Compute the p center nodes i.e. the p hubs k with the smallest distances $\sum_{j=1}^{N} d_{kj}$ to other nodes and locate the p hubs at these p nodes.
- Allocate each non hub node to its nearest hub.

Note that other heuristics for generating initial feasible solutions are proposed in the literature depending on the nature of the problem studied [23].

We use *randon_generate*(s) that randomly permutes in the solution s one hub with one non hub and allocates each non-hub to its nearest hub to generate a new individual

(solution) from s. We apply R times *randon_generate*() to the intial solution P_p, generated as mentionned above, to get R initial feasible solutions $P_p^i, 0 \leq i < R$. Then we run in parallel $GA(P_p^0), \ldots GA(P_p^{R-1})$ to get the solution of the USApHMP; here $GA(P_p^i)$ is the parallel GA with initial solution P_p^i.

2.4.1 Genetic Operators

Given a current solution s, the genetic operators used here consist of:

- *Generate*(s) which generates p_1 and p_2 from the current solution s by computing $p_1 = random\_generate(s)$ and $p_2 = random\_generate(s)$.
- *Crossover*(p_1, p_2) which creates two children ch_1 and ch_2 from (p_1, p_2) by performing a random cut in p_1 and in p_2, then copying the first part of p_1 to ch_1 and the first part of p_2 to ch_2, interchanging for both p_1 and p_2 the offsprings informations.
- *Correction*(s) which performes a correction to ensure the validity of the solution s, in terms of the number of hubs if different of p. If the number of hubs is greater than p then keep the p hubs with the greatest $O_i + D_i$. If the number of hubs is less than p then the missing hubs are randomly chosen among the non-hubs with the greatest amount of flow $O_i + D_i$. We apply the *correction*() operator to p_1 and p_2 to get c_1 and c_2 each with exactly p hubs (located at nodes where the traffic density is).
- *Mutation*() whose objective is to enlarge the search space and to avoid the convergence to the same solution. The mutation operator used here consists of permuting one hub with its neighbor non-hub.

2.4.2 Solution Evaluation

The following definitions of fitness are used in the standard benchmarks to evaluate the solutions. Let $A = \sum_i \sum_k d_{ik} z_{ik} \chi O_i + \gamma D_i + \sum_i \sum_k \sum_l \alpha d_{kl} y_i^{kl}, B = \sum_i \sum_j w_{ij}$, and $C = \sum_k F_k z_{kk}$.

- For the USAHLP: The fitness for CAB data is given by $\frac{A}{B} + C$ and the fitness for all other data instances except PlanetLab is given by $10^{-3}(A + C)$.
- For the USApHMP: The fitness for CAB data is given by $\frac{A}{B}$ and the fitness for all other data instances except PlanetLab is given by $10^{-3}A$.

Note that the reason of multiplying by 10^{-3} is to obtain the unit cost for flow transportation. This was discovered when we tried to reproduce the optimal solutions and this was confirmed by contacting M.R. Silva [14].

2.5 The GPU Implementation

Graphics Processing Units (GPUs) are available in most personal computers. They are used to accelerate the execution of a variety of problems. The smallest unit in the GPU that can be executed is called a thread. Threads (all executing the same code and synchronized) are grouped into blocks of equal size that are grouped in grids (blocks are independent and cannot be synchronized).

The memory hierarchy of the GPU consists of three levels: 1) the global memory that is accessible by all threads 2) the shared memory accessible by all threads of a block and 3) the local memory (register) accessible by a thread. Shared memory has a low latency (2 cycles) and is of limited size. Global memory has a high latency (400 cycles) and is of large size (4 GB for the Quadro).

An entire block is assigned to a single SM (Stream Multiprocessor). Each SM is composed of 32 streaming processors that share a same shared memory. Several blocks can run on the same SM. Each block is divided into Warps (32 threads by Warp) that are executed in parallel. The programmer must control the block sizes, the number of Warps and the different memory access.

A typical CUDA program is a C program where the functions are distinguished based on whether they are meant for execution on the CPU or on the GPU. Those executed on the GPU are called kernels and are executed by several threads. We implemented the previously presented algorithm on GPU (Nvidia Quadro with 4 GB and 384 cores running under CUDA 7.5 environment) and compare it to the best-known results in terms of computational time and quality of solutions.

2.5.1 The GPU Implementation of the GA

The GA creates R subpopulations each with n individuals ($R < 100$ for our implementation). So, the GPU is partitioned into R blocks; each with n threads (we define the master thread of each block as the thread 0 and the global master thread as the thread 0 of $block_0$). The matrices W and D are stored in the global memory. The $block_i$, $0 \leq i < R$ stores in its shared memory the genetic code of the initial solution P_p^i and executes the GA with P_p^i as initial solution as follow. At the beginning P_p (generated as indicated in Sect. 2.4) is duplicated in all $block_i$ and assigned to P_p^i, $0 \leq i < R$. Let $T_0^i, \ldots, T_{n-1}^i$ be the threads of the $block_i$. Each thread T_j^i generates a new solution (individual) $p_j^i = random\_generate(P_p^i)$. Then $(p_0^i, \ldots, p_{n-1}^i)$ becomes the initial subpopulation of the GA executed by the $block_i$. Note that the subpopulation size n is the same for all blocks. Each thread T_{2j}^i generates two children, namely ch_1 and ch_2 by crossing the parents (p_{2j}^i, p_{2j+1}^i) then T_{2j}^i applies the correction to ch_1 to obtain a new feasible individual (say c_{2j}) and T_{2j+1}^i applies the correction to ch_2 to obtain a new feasible individual (say c_{2j+1}). To enlarge the search space (to avoide local optimum), we use the mutation operator with a probability equal to 10^{-5} to the subpopulation $(c_0^i, \ldots, c_{n-1}^i)$ which consists of exchanging one hub with its

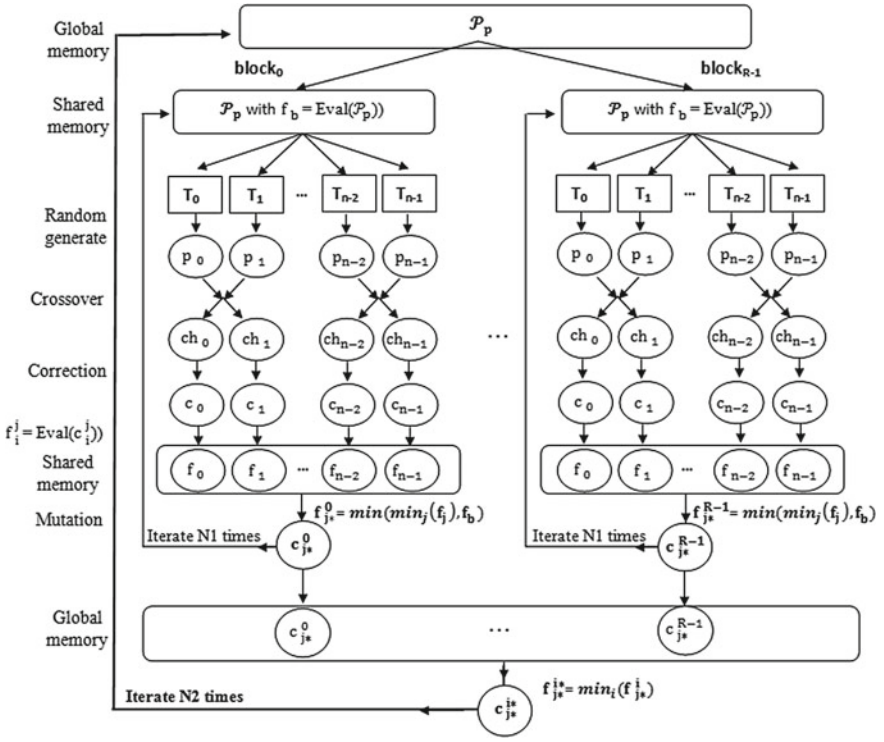

Fig. 2.2 The parallel GA for the USApHMP

neighbor non-hub. Next, each T_j^i allocates the non hubs of c_j^i to their nearest hubs and computes $f_j^i = fitness(c_j^i)$.

All c_j^i and f_j^i are stored in the shared memory of the $block_i$. Therefore, the master thread of the $block_i$ selects the individual c_{j*}^i with $min(min_j(f_j^i), f_b^i)$ where f_b^i is the cost of P_p^i (the initial solution) and updates the ancestor P_p^i as c_{j*}^i for the next iteration if any. This inner-loop of GA terminates after $N1$ iterations (the same for all the blocks).

The c_j, $0 \leq i < R$, are asynchronously computed and copied in the global memory and the individual c_{j*}^{i*} with $min_i(f_{j*}^i)$ is selected as the final solution or as the new value of the ancestor P_p for the next iteration, if any, of the outer-loop. The process is repeated $N2$ times. This implementation is summarized in the Fig. 2.2 and the pseudo CUDA code executed by the CPU is the following.

1. Generate(P_p) and copy it in the global memory of the GPU;
2. Define the blocks and the grid as: dim3 dimBlock(n,1); dim3 dimGrid(R,1);
3. Launch the kernel $GA(P_p)$: GA<<< $dimGrid, dimBlock$ >>> (P_p);

The R blocks create R subpopulations each with n individuals to find a solution for the USApHMP. Therefore, $N \times R$ blocks are needed to compute the N solutions for the USApHMP ($p = 1, \ldots, N$) to get the solution of the USAHLP.

2.6 Computational Results

We tested our implementation on CAB, AP, PlanetLab and Urand datasets.

- The CAB dataset of instances, introduced in [6] is based on airline passenger flow between 25 US cities. It contains distances that satisfy the triangle inequality and a symmetric flow matrix between cities. The instances are of 10, 15, 20 and 25 nodes. The distribution and collection factors γ and χ are equal to 1, while the discount factor α takes the values 0.2, 0.4, 0.6, 0.8 or 1 and the fixed cost for establishing hubs (F_k) is 10, 150, 200, or 250.
- The AP (Australian Post) dataset is a real-world dataset representing mail flows in Australia. For all instances, the parameters are $\alpha = 0.75,\ \gamma = 3,\ \chi = 2$. The mail flows are not symmetric and there are possible flows between each node and itself. The fixed costs F_k are different. For all problem sizes, there are two fixed costs for hubs Tight and Loose costs. The instances with tight costs are mostly harder to solve, because nodes with larger total flows are set with higher fixed costs, because nodes with larger total flows are set with higher fixed costs.
- The Urand dataset are random instances up to 400 nodes generated by Meyer et al. [32], while the instance with 1000 nodes was generated by [15]. In these benchmarks, the nodes coordinates were randomly generated from 0 to 10^5 and the flow matrix was randomly generated.
- The PlanetLab instances are node-to-node delay data for performing internet measurements [15]. In these networks, $\alpha = \gamma = \chi = 1$ and the distance matrix does not respect the triangle inequality due to missing links (i.e. firewall restrictions). The flow w_ij is equal to 0 if $i = j$ and 1 otherwise.

We report the results for the four datasets presented above. We compare our results to the recent work of [22] in terms of computing time for USAHLP and to the results of [15] for the USApHMP. For our tests, the GA was executed 10 times on each instance. The same solution was obtained for the multiple executions of each instance. The time transfer between the CPU and GPU varies according to the number of nodes. The given times in all tables include the time of generating the initial solutions, the time of data transfers CPU-GPU and the computation of the solutions on the GPU.

The following notations are used in Tables 2.1, 2.2, 2.3, 2.4, 2.5 and 2.6. N is the number of nodes in the instance. *BestSol* is the best solution if known else is written. *GpuSol* is the solution obtained by the GPU with opt if the solution is the optimum. *Topt* is the best time (in seconds) in the literature. $TGpu$ is the time (in seconds) for our implementation. p the number of hubs. Tight (T) and Loose (L).The new results are shown in these tables in bold type.

All instances of CAB are solved to optimality in times $< 1s$ for the USApHMP as in [14, 22]. So, we will report these results only for the USAHLP.

Tables 2.1, 2.2, 2.3, 2.4 and 2.5 give a comparison of our results to the best results obtained in the literature to solve USApHMP in all these benchmarks. As shown in Table 2.1 for the AP instances we obtained optimal solutions for 100 and 200 nodes

Table 2.1 Solutions of the USApHMP on AP instances

N	p	BestSol	GpuSol	TGpu
100	5	136929.444	Opt	1.310
	10	106469.566	Opt	1.310
	15	90533.523	Opt	1.49
	20	80270.962	Opt	1.63
200	5	140062.647	Opt	3.602
	10	110147.657	Opt	3.722
	15	94459.201	Opt	3.783
	20	84955.328	Opt	3.841
300	5	–	**174914.73**	5.631
	10	–	**134773.55**	5.711
	15	–	**114969.85**	5.896
	20	–	**103746.44**	5.876
400	5	–	**176357.92**	6.741
	10	–	**136378.19**	6.846
	15	–	**117347.10**	7.102
	20	–	**104668.27**	7.423

Table 2.2 Solutions of the USApHMP on PlanetLab instances

Instance	N	p	BestSol	GpuSol	TBest	TGpu
01-2005	127	12	2927946	**2904434**	148.9	0.5
02-2005	321	19	18579238	**18329984**	462.7	6.9
03-2005	324	18	20569390	**20284132**	543.8	7.5
04-2005	70	9	739954	**730810**	0.6	0.3
05-2005	374	20	25696352	**25583240**	622.6	8.3
06-2005	365	20	22214156	**22191592**	581.7	7.9
07-2005	380	20	30984986	**30782956**	546.6	8.5
08-2005	402	21	30878576	**30636170**	637.6	8.7
09-2005	419	21	32959078	**32649752**	684.9	9.3
10-2005	414	21	32836162	**32687796**	731.9	9.1
11-2005	407	21	27787880	**27644374**	588.3	9.2
12-2005	414	21	28462348	**28213748**	680.3	9.1

in a very short time (for nodes up to 50, we obtained the optimal solutions in time < 0.3 s). To our knowledge, the results on AP data instances for the pHMP with 300 and 400 nodes have not hitherto been reported in the literature. Hence, we claim that our results are the best solutions for these instances. We report our results for PlanetLab instances in Table 2.2. We can see that our approach outperforms those of [15] both in terms of cost and computing time. The state-of-the-art solutions given

Table 2.3 Solutions of the USApHMP on PlanetLab instances for other values of p

N	p	GpuSol	TGpu	N	p	GpuSol	TGpu
127	2	**3523236**	0.4	370	2	**812646**	0.2
	3	**3126396**	0.4		3	**784390**	0.2
	4	**3042004**	0.4		4	**758676**	0.2
	5	**2994190**	0.4		5	**752170**	0.2
	10	**2911194**	0.5		10	**728174**	0.3
	15	**3220752**	0.5		15	**719762**	0.3
	20	**2886548**	0.7		20	**713926**	0.3
321	2	**22060080**	5.9	374	2	**30871424**	8.0
	3	**21198104**	6.1		3	**29430588**	8.0
	4	**20364616**	6.1		4	**28198958**	8.1
	5	**19807670**	6.2		5	**27352248**	8.2
	10	**18981110**	6.7		10	**26214866**	8.2
	15	**18614864**	6.8		15	**25760530**	8.3
	20	**18423908**	7.0		20	**26802004**	8.3
324	2	**24139848**	7.1	365	2	**26192564**	7.5
	3	**23193758**	7.2		3	**25416392**	7.5
	4	**22536298**	7.2		4	**24528130**	7.6
	5	**22159678**	7.3		5	**23977334**	7.6
	10	**21339900**	7.4		10	**22708070**	7.7
	15	**20607754**	7.5		15	**22398924**	7.8
	20	**20672764**	7.6		20	**22190348**	7.9

Table 2.4 Solutions of the USApHMP on Urand large instances

N	p	BestSol	GpuSol	Topt	TGpu
1000	2	198071412.53	**8184986.50**	1.7245	9.321
	3	169450816.35	**7024184.00**	8.1550	9.785
	4	150733606.87	**6184749.01**	2.2240	10.431
	5	142450250.26	**5860994.06**	58.6070	10.89
	10	114220373.07	**4752317.00**	187.8385	13.7
	15	–	**4228256.88**	–	15.23
	20	198071412.53	**3928617.48**	403.4280	17.923

in [15] report results for couples of (N, p) with $p \approx \sqrt{N}$. We report in Table 2.3 the results for other possible values of p.

For the Urand instances, our GPU implementation gives the best solutions for instances with up to 400 nodes and outperforms those of [15] for instances with 1000 nodes. Concerning the computing times, our approach is very faster and gives the solutions in times $< 18s$ for all instances, while the best-known time is more than 7 minutes. Moreover, it increases rapidly with p in [15] whereas it does not change

Table 2.5 Solutions of the USApHMP on larger Urand instances generated by us

N	p	GpuSol	TGpu	N	p	GpuSol	TGpu
1500	20	454787506	196	4000	20	**3234999192**	3076
	30	**407155164**	286		30	**2983891783**	3276
	40	**380114045**	423		40	**2769550514**	3365
	50	**363586538**	574		50	**2644606684**	3648
2000	20	**805749722**	477	5000	20	**5085803132**	4662
	30	**733375448**	580		30	**4656787498**	4720
	40	**686515363**	714		40	**4353561395**	4996
	50	**655938000**	965		50	**4143849388**	5112
3000	20	**1804950952**	1157	6000	20	**7398401957**	5614
	30	**1642145354**	1544		30	**6675723961**	5748
	40	**1538548764**	1869		40	**6293053841**	5964
	50	**1468780124**	2086		50	**5999780197**	6212

Table 2.6 Solutions of the USAHLP on AP instances

N	Fi-type	BestSol	Hubs of BestSol	GpuSol	Hubs of GpuSol	TGpu
10	L	224250.05	3, 4, 7	224250.05	3, 4, 7	0.024
20	L	234690.95	7, 14	234690.95	7, 14	0.061
25	L	236650.62	8, 18	236650.62	8, 18	0.069
40	L	240986.23	14, 28	240986.23	14, 28	0.243
50	L	237421.98	15, 36	237421.98	15, 36	0.490
100	L	238016.28	29, 73	238016.28	29, 73	1.624
200	L	233803.02	–	**233801.35**	**43, 148**	3.316
300	L	263913.15	26, 79, 170, 251, 287	**258823.42**	**79, 126 , 170, 192, 287**	8.183
400	L	267873.65	99, 180, 303, 336	**259416.31**	**146, 303, 336**	13.32
10	T	263399.94	4, 5, 10	263399.94	4, 5, 10	0.026
20	T	271128.18	7, 19	271128.18	7, 19	0.072
25	T	295667.84	13	295667.84	13	0.085
40	T	293164.83	19	293164.83	19	0.297
50	T	300420.98	24	300420.96	24	0.548
100	T	305097.96	52	305097.93	52	1.864
200	T	272237.78	–	**272188.10**	**54, 122**	4.961
300	T	276023.35	30, 154, 190	**266030.76**	**30, 154, 190**	9.742
400	T	284037.25	101, 179, 372	**275769.09**	**101, 179, 372**	15.98

much for the GPU implementation. Clearly, our solutions are far better in term of cost than those of [15] as can be seen in Table 2.4, $GpuSol \approx \frac{BestSol}{24}$.

We report in Table 2.5 results for larger instances that we have generated using the same generation procedure as for the Urand instances [32]. These new challenging instances consist of large instances of up to 6000 nodes that have not hitherto been solved.

We solved the USAHLP on all 80 CAB data instances introduced in the literature, in execution time <0.04 s. In particular, we confirm the value of 1081.05 found in [22] for $N = 10, \alpha = 1, F_k = 150$, whose value is incorrect in certain works [14]. On the other hand, the results for the standard AP data are reported in Table 2.6. We obtained optimal solutions up to 200 nodes for the two types of fixed cost for hubs (T) and (L). Our approach widely surpasses the best-known solution [14] for the 200, 300 and 400 node instances and gives solutions for unsolved instances.

2.7 Conclusion

We implemented a parallel genetic algorithm on GPU for solving the Uncapacitated Single Allocation p-Hub Median Problem. We used this implementation as basic kernel to solve the Uncapacitated Single Allocation Hub Location Problem. We showed the effectiveness of our implementation on well-known benchmarks. Indeed, our approach improves the best-known solutions in cost and in computing times for well-known benchmark instances up to 1000 nodes. Moreover, we solved problems instances up to 6000 nodes so far unsolved.

Our futur works concern the design and implementation of an exact parallel tree-based algorithm to solve the studied hub problems as these algorithm structures seem to be suitable for parallel computers. Other developments concern the extension of this work to other versions of the hub location problem.

Acknowledgements We thank Dr. M. OKelly, M.R. Silva, A. Ilic' and R. Abyazi-Sani for the instance datasets provided. This work was supported by the FEDER CLASSE2 Project.

References

1. Campbell, J. F. (1994). Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, *72*(2), 387–405.
2. Campbell, J. F., Ernst, A. T., & Krishnamoorthy, M. (2005). Hub arc location problems: part i, introduction and results. *Management Science*, *51*(10), 1540–1555.
3. Contreras, I., Cordeau, J. F., & Laporte, G. (2011). The dynamic uncapacitated hub location problem. *Transportation Science*, *45*(1), 18–32.
4. de Camargo, R. S., & Miranda, G. (2012). Single allocation hub location problem under congestion: Network owner and user perspectives. *Expert Systems with Applications*, *39*(3), 3385–3391.

5. Campbell, J. F., & O'Kelly, M. E. (2012). Twenty-five years of hub location research. *Transportation Science*, *46*(2), 153–169.
6. O'kelly, M. E. (1987). A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, *32*(3), 393–404.
7. Sohn, J., & Park, S. (1998). Efficient solution procedure and reduced size formulations for p-hub location problems. *European Journal of Operational Research*, *108*(1), 118–126.
8. Abdinnour-Helm, S. (1998). A hybrid heuristic for the uncapacitated hub location problem. *European Journal of Operational Research*, *106*(2), 489–499.
9. Ernst, A. T., & Krishnamoorthy, M. (1998). Exact and heuristic algorithms for the uncapacitated multiple allocation p-hub median problem. *European Journal of Operational Research*, *104*(1), 100–112.
10. Bryan, D. (1998). Extensions to the hub location problem: Formulations and numerical examples. *Geographical Analysis*, *30*(4), 315–330.
11. Horner, M. W., & O'Kelly, M. E. (2001). Embedding economies of scale concepts for hub network design. *Journal of Transport Geography*, *9*(4), 255–265.
12. Labbe, M., Yaman, H., & Gourdin, E. (2005). A branch and cut algorithm for hub location problems with single assignment. *Mathematical Programming*, *102*(2), 371–405.
13. Chen, J. F. (2007). A hybrid heuristic for the uncapacitated single allocation hub location problem. *Omega*, *35*(2), 211–220.
14. Silva, M. R., & Cunha, C. B. (2009). New simple and efficient heuristics for the uncapacitated single allocation hub location problem. *Computers and Operations Research*, *36*(12), 3152–3165.
15. Ilic, A., et al. (2010). A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, *206*(2), 289–300.
16. Maric, M., Stanimirovic, Z., & Stanojevic, P. (2013). An efficient memetic algorithm for the uncapacitated single allocation hub location problem. *Soft Computing*, *17*(3), 445–466.
17. Bailey, A., Ornbuki-Berrnan, B., & Asobiela, S. (2013). Discrete pso for the uncapacitated single allocation hub location problem. In *Computational Intelligence in Production and Logistics Systems (CIPLS)* (pp. 92–98).
18. Damgacioglu, H., Dinler, D., Ozdemirel, N. E., & Iyigun, C. (2015). A genetic algorithm for the uncapacitated single allocation planar hub location problem. *Computers and Operations Research*, *62*, 224–236.
19. Ting, C. J., & Wang, H. J. (2014). A threshold accepting algorithm for the uncapacitated single allocation hub location problem. *Journal of the Chinese Institute of Engineers*, *37*(3), 300–312.
20. Meier, J. F., & Clausen, U. (2015). Solving classical and new single allocation hub location problems on Euclidean data. In *Optimisation Online*, 03-4816.
21. Rostami, B., et al. (2015). Lower bounding procedures for the single allocation hub location problem. In *Electronic notes in discrete mathematics* (Vol. 320).
22. Abyazi-Sani, R., & Ghanbari, R. (2016). An efficient tabu search for solving the uncapacitated single allocation hub location problem. *Computers and Industrial Engineering*, *93*, 99–109.
23. Shobeiri, A. (2015). Grasp metaheuristic for multiple allocation p-hub location problem. Ph.D. thesis, Concordia University, Montreal, Canada
24. Kratica, J., et al. (2012). Genetic algorithm for solving uncapacitated multiple allocation hub location problem. *Computing and Informatics*, *24*(4), 415–426.
25. Topcuoglu, H., et al. (2005). Solving the uncapacitated hub location problem using genetic algorithms. *Computers and Operations Research*, *32*(4), 967–984.
26. Benaini, A., & Berrajaa, A. (2017, January). Gpu based algoithm for the capacitated single allocation hub location problem. Submitted for publication.
27. Pospichal, P., Jaros, J., & Schwarz, J. (2010). Parallel genetic algorithm on the cuda architecture. In *Applications of evolutionary computation* (pp. 442–451). Berlin: Springer.
28. Talbi, E. G. (2013). Metaheuristics on gpu. *Journal of Parallel Distributed Computing*, *73*(1), 1–3.

29. Luong, T. V., Melab, N., & Talbi, E. G. (2013). Gpu computing for parallel local search metaheuristic algorithms. *IEEE Transactions on Computers*, *62*(1), 173–185.
30. Ernst, A. T., & Krishnamoorthy, M. (1996). Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, *4*(3), 139–154.
31. Naeem, M., & Ombuki-Berman, B. (2010). An efficient genetic algorithm for the uncapacitated single allocation hub location problem. In *IEEE Congress on Evolutionary Computation*.
32. Meyer, T., Ernst, A. T., & Krishnamoorthy, M. (2009). A 2-phase algorithm for solving the single allocation p-hub center problem. *Computers and Operations Research*, *36*(12), 31433151.