Journal of Computer Science

Original Research Paper

A GPU-Based Genetic Algorithm for the Multiple Allocation P-Hub Median Problem

[1]Achraf Berrajaa, [2]Ayyoub El Outmani and [3]Abdelhamid Benaini

[1]Department of Computer Science, Euromed Research Center, Euromed University of Fes, Morocco
[2]Department of Computer Science, LARI, Faculty of Sciences Oujda, Mohammed Premier University, Oujda, Morocco
[3]Department of Mathematics, LMAH, Faculty of Sciences and Techniques, Normandie University, le Havre, France

Article history
Received: 20-08-2022
Revised: 07-02-2023
Accepted: 13-02-2023

Corresponding Author:
Achraf Berrajaa
Department of Computer Science, Euromed Research Center, Euromed University of Fes, Morocco
Email: berrajaa.achraf@gmail.com

Abstract: As the sizes of realistic hub location problems increase as time goes on (reaching thousands of nodes currently) this makes such problems difficult to solve in a reasonable time using conventional computers. This study aims to show that such problems may be solved in a short computing time and with high-quality solutions using the computational power of the GPU (actually available in most personal computers). So, we present a GPU-based approach for the uncapacitated multiple allocations p-hub median problems. Our method identifies the nodes that are likely to be hubs in the optimal solution and improves them via a parallel genetic algorithm. The obtained GPU implementation reached within seconds the optimal or the best solutions for all the known benchmarks we had access to and solved larger instances up to 6000 nodes so far unsolved. Compared to this study, no other article dealing with hub location problems has presented results for instances as large.

Keywords: Genetic Algorithm, P-Hub Median Problem, Multiple Allocations, GPU

Introduction

Hub location models appear in various fields such as transportation, telecommunications, etc., and have a wide range of applications in air or ground transportation networks, postal delivery systems, and so on. In these models, traffic from origin to destination is not sent directly but must be routed through a specific set of nodes called hubs. Every other non-hub node is assigned to a single hub (single allocation) or to multiple hubs (multiple allocations). Assuming (i) Open hubs are fully connected to more efficient paths, allowing a discount factor α to be applied to transport costs between hubs. (ii) The triangle inequality applies to the distance between nodes (iii). There is no direct connection between the two non-hub nodes. Thus, the traffic flow will flow through one or two nodes, and economies of scale are achieved by consolidating traffic at the hubs. More precisely, the transmission cost of flow w_{ij} from i to j via hubs k and l is given by $C_{ijkl} = (\chi dik + \alpha dkl + \gamma d_{lj})w_{ij}$ where d_{ik}, d_{kl}, and d_{lj} are respectively the distances from i to k, from k to l and from l to j, χ is the collection cost per unit, γ is the distribution cost per unit, where $\alpha < \chi$ and $\alpha < \gamma$.

In the multiple allocation hub location problems, a non-hub node is allowed to be assigned to more than one hub, depending on the destination of the flows originating from that node. The goal is to locate p hubs and map non-hubs (spook) to hubs so that the overall shipping cost is minimized. Given the number p of hubs a priori, the problem is called the p-hub median problem. Also, if the hubs have no capacity constraints, the problem is known as the Uncapacitated Multiple Allocation p-Hub Median Problem (UMApHMP). Multiple allocations increase the flexibility of the model and it is expected to have lower-cost solutions compared to the single allocation case.

Since the UMApHMP belongs to the class of NP-hard problems, it is difficult for exact optimization methods to solve large benchmarks (for instance, the hub and spook network of the China Deppon logistics includes 50 hubs and 5400 nodes and these numbers increase as time goes by Wang and Huang (2016). Therefore, heuristic and parallel computing methods are promising approaches for solving real problems with large sizes (Benaini *et al*., 2022). So, we propose a genetic algorithm parallelized for the GPU for solving the UMApHMP. To our knowledge, this is the first GPU-based approach dedicated to this problem. Our Genetic Algorithm (GA) starts from different initial solutions and improves them via classical genetic operators. The initial solutions locate the hubs which are likely to be hubs in the optimal solution. Consequently, they greatly enhance the convergence of the GA. We present this approach and we show the efficiency of our GA under experimental results on well-known

benchmarks and on large random instances of up to 6000 nodes generated by us.

Several efficient heuristics have been implemented for both single and multiple allocations p-hub median problems. Among the most promising algorithms are Tabu Search (TS), Genetic Algorithm (GA), scatter search, simulated annealing, greedy randomized adaptive search, and electromagnetism like methods. A brief presentation of the corresponding references is below.

Sangsawang (2011) proposed a multiple tabu search for solving the UMApHMP based on the convex hull technique to identify the nodes that send/receive flows to multiple hubs. He uses an $n \times 2n$ array to identify the intermediate hubs between origin-destination pairs. Rabbani and Kazemi (2015) presented a GA optimization approach for the uncapacitated multiple allocation p-hub center problem. He compared the nearest hub and all pair shortest path strategies for multiple allocations on the CAB and the AP data set. Zhang *et al*. (2023), the authors used Variable Neighborhood Search (VNS) heuristics to solve this problem. Shobeiri (2015) presented greedy deterministic and randomized algorithms for constructing an initial feasible solution and improving it by using local improvement techniques. He studies the impact of the quality of the initial feasible solution on the local improvement phase of AP data instances. Sender and Clausen (2013) developed different versions of a local search approach for solving the capacitated version of the problem derived from a practical application in the network design of German wagonload traffic. Boland *et al*. (2004) observe characteristics of optimal solutions for three versions of the multiple allocation problems. Then, they used these characteristics to develop preprocessing techniques and tightening constraints and apply them to the appropriate problems. Chen (2006) gives an approach to derive an upper bound for the optimal number of hubs for the Uncapacitated Multiple Allocation Hub Location Problem (UMAHLP). He uses this upper bound with the simulated annealing method to solve this problem. Zetina *et al*. (2017) study cases where the demand and the transportation cost are subject to interval uncertainty. They present mixed integer programming formulations for these problems and computational results with a general-purpose solver. For a robust optimization, Benaini and Berrajaa (2020) propose a genetic algorithm to solve the robust uncapacitated single allocation and also Rouzpeykar *et al*. (2022) developed a genetic algorithm to solve p-hub location and revenue management problem.

Yaman (2011), introduces the R-allocation p-hub median problem, where each node must be allocated to at most r hubs. This problem generalizes two versions of the p-hub median problem, single and multiple allocations ($r = 1$ and $r = p$). His computational study shows that single-allocation solutions can be significantly more expensive than multiple-allocation solutions. However, assigning one node to another hub or two can save significant routing costs, and the resulting network may be easier to manage. Kratica (2013) introduced the original electromagnetism like metaheuristic with a scaling technique. Results on several benchmarks and on large instances up to 1000 nodes showed high performance of the proposed method.

Several Genetic Algorithms (GA) are proposed to solve different variants of the hub location problems. Most of them use binary/integer arrays as encoding. A simple and powerful GA for the multiple allocation problem was proposed by Kratica *et al*. (2005). This GA uses binary N-string encoding and original genetic operators. The author proposed caching to reduce the computational time in the evaluation function and shows the effectiveness of the GA through tests on CAB and AP instances up to 200 nodes. We adopt the binary N-string encoding for our GA. However, the particularity of the UMApHMP is that once the set of hubs is defined, an optimal allocation of non-hubs to hubs is obtained by the shortest paths rule. So, given the N-string encoding, it is necessary to compute all pair shortest paths to fully determine the solution and to evaluate its fitness which is expensive computationally. To avoid re-computing all the paths at each iteration of the GA, we update those of the current solution after each modification of the latter. So, we must indicate how to represent all the shortest paths of a solution and how to update them after a change in the solution.

Our GA starts from an initial solution that locates hubs at the nodes that are likely to be hubs in an optimal solution, in order to accelerate the convergence to the optimal or best solution. It is therefore necessary to identify potential optimal hubs. For this purpose, we are inspired by the work of Peker *et al*. (2016) on the analysis of optimal hub locations for single allocation problems and by other researchers on this subject due to Demir *et al*., (2022); Alvarez Fernandez *et al*. (2022); Contreras and O'Kelly (2019); Shobeiri (2015).

Finally, it should be mentioned that very few GPU solutions for the hub location problems are proposed in the literature. Benaini *et al*. (2019) proposed a GPU implementation for solving the single allocation phub median problem. Lim and Ma (2013) presented a GPU-based parallel vertex substitution algorithm for the p-hub median problem and Santos *et al*. (2010) proposed a GPU implementation of the GRASP metaheuristic for the p-hub median problem. More recently, AlBdaiwi and AboElFotoh (2017) presented a new genetic algorithm based on a pseudo-Boolean formulation of the p-median problem that he implemented on GPU. Interesting experimental tests on hard instances of sizes up to 900 are reported. We would like to highlight that, compared to our work, no article dealing with the hub location problems, proposed results for instances as large as 6000 nodes. In

view of these related works, we can affirm that the contributions of this study are in the method of locating optimal hubs and in the GPU-based approach for solving the UMApHMP.

Problem Statement and Formulation

The UMApHMP can be informally stated as follow. Given a set N of n nodes, find a sub-set of p hubs $H \subset N$ that minimizes:

$$\sum_{i,j \in N} \min_{k,l \in H} C_{ijkl}\, x_{ijkl}$$

where the decision variable $x_{ijkl} = l$ if the flow w_{ij} from i to j goes via hubs k and l (k may be equal to l) and 0 otherwise. Given that there are no capacity constraints on the hubs or links of the network, there always exists an optimal solution in which each flow w_{ij} fully routed on a single path (Contreras and O'Kelly, 2019) which justifies the fact that the x_{ijkl} are binary variables. The general constraints imposed by the UMApHMP are the following p is the number of hubs. It is assumed that transportation costs and traffic flow are known and determined. The traffic between a pair of nodes i and j must be routed through either one or two established hubs k and l. All hubs are interconnected and none of the non-hubs are directly connected. Each non-hub can be allocated to multiple hubs. We call the hubs of optimal or best solution optimal hubs. A more precise formulation of the UMApHMP can be found in (Boland *et al*., 2004; Kratica *et al*., 2005).

Figure 1(a) shows an example of a hub network with $n = 5$ nodes. The number of located hubs is $p = 2$. Figure 1(b) presents a solution for the single allocation case with hubs located at nodes 2 and 4. The non-hubs 1 and 5 are allocated to hub 4 and non-hub 3 to hub 2. For instance, the flow from 1-3 takes path 1-4-2-3. A multiple allocation solution is presented in Fig. 1(c) with hubs located at nodes 2 and 4. The non-hub 3 is allocated to hubs 2 and 4, non-hub 1 is allocated to hubs 2 and 4, and non-hub 5 is allocated to hub 4. The flow from 1 to 3 takes the path having the minimum cost among the three paths 1-4-2-3 or 1-2-3 or 1-4-3.

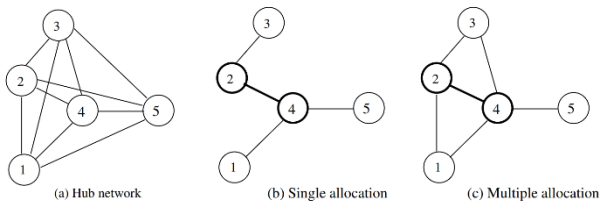

Fig. 1: An example of p-HMP with solutions for single and multiple allocations

Materials

This study is implemented on GPU Nvidia Quadro with 2 GB and 2 SM and 384 cores running under CUDA 11.0 environment.

Methods

Location of Initial Hubs

The authors are inspired by the work of Peker *et al*. (2016), that have an interesting analysis of the optimal hub locations in the case of a single allocation. Using the optimal solutions for small CAB and AP instances, they observe that the spatial distribution of nodes (betweenness and centrality) and the size of node requirements are the most influential parameters for locating optimal hubs. They proposed a CBS algorithm that partitions the nodes into clusters each centered at a node that is likely to be an optimal hub. We adopted the same methodology to conceive another simple and efficient algorithm to partition the set of nodes into p disjoint clusters each likely to contain an optimal hub for the UMApHMP. We showed the effectiveness of our approach by tests on several known benchmarks (including large random instances). The result is that the percentage of the number of clusters each containing at least one optimal hub is about 82%. Our clustering algorithm is based on the following two observations: (i) In an optimal solution of the UMApHMP, a non-hub node is necessarily allocated to the hub closest to it (ii) A node k with the greatest index $I(k) = O_k + D_k$ is a potential optimal hub and the closest nodes to it are unlikely to be also optimal hubs, they are rather the nodes allocated to hub k. The following simple algorithm partitions the set of nodes into p disjoint clusters C_i each likely to contain a hub of an optimal or best solution.

Clustering algorithm:

L = the list of nodes sorted in decreasing order according to their indexes $I()$;
$r = \lceil N/p \rceil$; $i = 1$;
while ($i < p$) **do**
  h_i = is the first node of L not assigned to $C_1 \cup \cdots \cup C_{i-1}$;
  if ($i < p$) then C_i = the set of the r nodes closest to h_i (including h_i);
  else C_i = the remaining nodes in L;
  Remove C_i from L ($L = L \backslash C_i$); $i = i+1$;
end while

Two remarks on this algorithm. First, as this algorithm is dedicated to the UMApHMP, the C_i may be non-disjoint and consequently may contain different numbers of nodes instead of r nodes. Secondly, the algorithm can be improved to take into account the isolated nodes as in Peker *et al*. (2016) where the generated clusters (circles) have the same radius and are centered at nodes selected among the $2p$ most important nodes. These centers are

located in hubs. Here, the clusters C_i, have a different radius and are centered at h_i, and each contains the same number of nodes (except for C_p). Identifying the potential optimal hub in each C_i is a more delicate task. Indeed, the potential hub in C_i is not necessarily h_i but may be a node close to h_i (according to some criteria). We tested different criteria to select the optimal hub in each cluster. We observed that the node $k \in C_i$ with small $\sum_{o,d \in C_i} \left(\chi d_{ok} + \gamma d_{kd}\right) w_{od}$ is the best candidate optimal hub in C_i for several benchmarks but not for all. We confess that we failed to define an efficient criterion, common to all tested benchmarks.

Figure 2 illustrates this on the AP20.5 instance with $n = 20$ and $p = 5$. The ordered list is $L = (14, 15, 2, 19, 6, 13, 20, 16, 3, 5, 9, 4, 12, 18, 10, 7, 8, 17, 1, 11)$. The five clusters are $C_1 = (14, 19, 15, 18)$, $C_2 = (2, 3, 1, 6)$, $C_3 = (13, 9, 10, 5)$, $C_4 = (20, 16, 11, 12)$, $C_5 = (4, 7, 8, 17)$ and are represented in Fig. 2 by different symbols (C_1 with stars, C_2 with circles, etc.,). The optimal hubs are {14, 2, 13, 12, 6} and are represented by filled symbols in this figure. Only C_5 does not contain an optimal hub. If we select h_i as a hub in C_i then we obtain the set of initial hubs {14, 2, 13, 20, 4}. Three of them are actually optimal hubs but not the other two.

The Genetic Algorithm

Most of the GAs for the hub location problems proposed in the literature use binary or integer arrays to encode the solutions. The operators mostly used in genetic algorithms for the UMApHLP consist in randomly exchanging some hubs with non-hub nodes regardless of the encoding used. Indeed, once the set of hubs is defined, an optimal allocation is obtained by the shortest path algorithm and therefore, operators that change the node allocations are useless.

Solution Representation and Encoding

We adopt a simple encoding by a binary string s of length n where $s(k) = 1$ denotes that k is an established hub and $s(k) = 0$ denotes that k is a non-hub node. We represent all shortest cost path i-k-l-j of a solution encoded by s by a two-dimensional array T_s of size $n \times 2n$; where a row represents the origin node, a column identifies the destination node, and the l-column and k-column represent the intermediate hubs. Sangsawang (2011) used this representation in a tabu search approach.

For instance, the representation T of the solution of Fig.1(c) is given in Table 1.

In this example, $T(3, 5) = (2, 4)$ means that the shortest cost path from node 1 to node 3 is 3-2-4-5, and $T(1, 5) = (4, 4)$ means that the shortest cost path from node 1 to node 5 is 1-4-5, etc. Note that if the l-column of $T(i, j) = $ j or the k-column of $T(j, i) = j$ for all i then j is hub. This allows the identification of hubs from T. The representation T_s is associated with the encoding s and defined by $T_s(i, j) = argmin_{k,1:\, s(k) = s(l) = 1} C_{i,jkl}$.

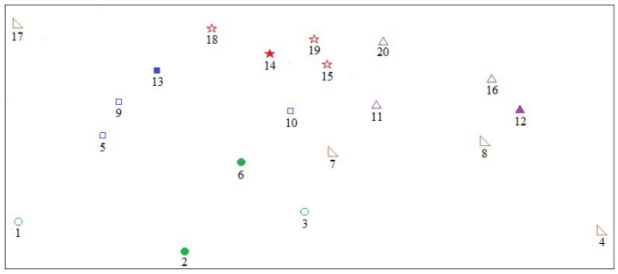

Fig. 2: Clustering and optimal hubs for the AP20.5 instance

The genetic operators operate on the N-string s and the representation T_s, which must be updated after the change of hubs of s, gives the fitness $\sum_{i,j} C_{ijT(i,j)}$ of s.

Genetic Operators

The following classical operators have been implemented in our GA.

- Given an individual s and its representation T_s, generate new individual s' from s by k exchange() consists in randomly choosing k hubs of s and exchanging them with k non-hubs then update T_s to get T_s'
- A population of R individuals is obtained from an initial individual s by applying R times k-exchange() to s
- The crossover with single random point crossover exchanges the two parts of parents p_1 and p_2 to generate two springs ch_1 and ch_2. If the number of ones in the code of ch_i, ($i = 1, 2$); is different from p then ch_i is corrected to feasible as follows. If the number of hubs of ch_i is $> p$ then keep the p hubs with greatest $I()$ else the missing hubs are randomly chosen from the non-hub nodes. This produces two feasible individuals c_1 and c_2. The representation T_{c_i} ($i = 1, 2$) is computed as follows. If the number of hubs common to ci and p_1 is greater than the number of hubs common to c_i and p_2 then update T_{p_1} else update T_{p_2} to get T_{c_i}
- The aim of the mutation is to enlarge the search space and to avoid the local optima. We use a mutation operator that changes the hubs for 2% of individuals randomly chosen and updates the representations T of the obtained individuals

Exchanging hubs with non-hubs and consequently, updating the representation T is the most expensive operation, in computing time, of this GA. Therefore, we designed the updating process to be as efficient as possible without re-computing all pair shortest paths.

Table 1: Example of *T* representation

	1		2		3		4		5	
j										
i	*k*	l	*k*	l	*k*	l	*k*	l	*k*	l
1	4	4	2	2	2	2	4	4	4	4
2	2	2	2	2	2	2	2	4	2	4
3	2	2	2	2	2	2	2	4	2	4
4	4	4	2	2	4	2	4	2	4	4
5	4	4	4	2	4	2	4	4	4	4

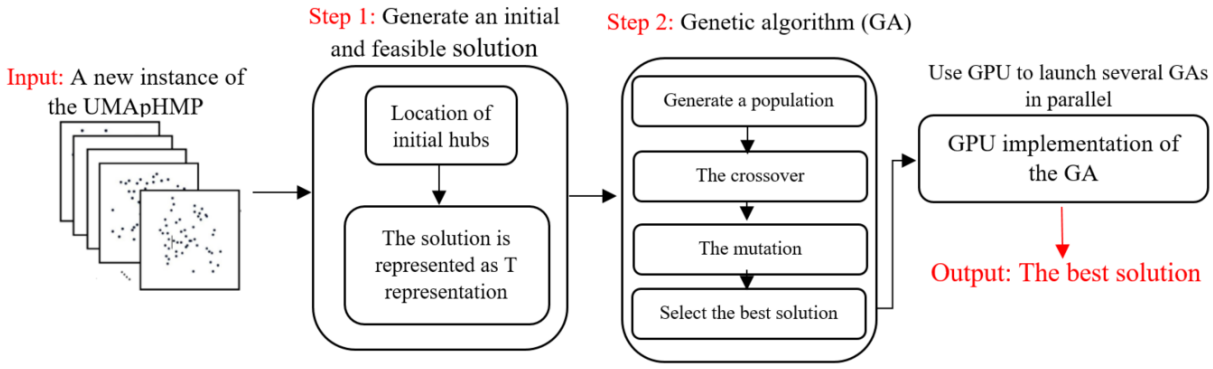

Fig. 3: The proposed algorithm steps

Updating the Representation T after the Change of Hubs

Let H be the set of hubs of a solution s represented by T_s. Then:

$$Ts(i,j)=argmin_{k,l\in H} C_{ijkl}, with$$
$$C_{ijkl}=(\chi d_{ik}+\alpha d_{kl}+\gamma d_{lj})W_{ij}$$

Let, s' the solution obtained by exchanging t hubs $h_1,\cdots,h_t$ of s with t non-hubs $h_1',\ldots,h'_t$ of s and $T_{s'}$ its represented. Let $H_1 = \{h_1,\ldots,h_t\}$, $H'_1 = \{h'_1,\ldots,h'_t\}$ and $H' = (H\backslash H_1) \cup H'_1$. Then H' becomes the set of bubs of s' and $T_s'(i, j) = argmin_{k,l\in H'} C_{ijkl}$. We show how to compute $T_s'(i, j)$ for all i, j. Assume that $j\in H'_1$ then any path (of s') from i to j passes through at most another hub.

Hence:

$$T_{S'}(i,j)=(argmin_{K\in H'}\left(\chi d_{ik}+\alpha d_{k,j}\right),j)$$

is done in $O(p)$ steps. Likewise, if $K \in H'$ then:

$$T_{S'}(i,j)=\left(i,argmin_{K\in H'}\left(\alpha d_{il}+\gamma d_{lj}\right)\right)$$

is done in $O(p)$ steps. Now, suppose that neither i nor j is a hub of s' and let $T_s(i, j) = (k_0, l_0)$. Then:

$$\min_{k,l\in H'} C_{ijkl} = \min \left(\min_{k,l\in H\backslash H1} C_{ijkl}, \min_{k\in H1',l\in H'} C_{ijkl}, \min_{k\in H',l\in H_1'} C_{ijkl}\right)$$

If neither k_0 nor l_0 is in H_1 then:

$$\min_{k,l\in H\backslash H_1} C_{ijkl} = \min_{k,l\in H} C_{ijkl}$$

Hence:

$$argmin_{k,l\in H\backslash H_1} C_{ijkl} = T_s(i,j)$$

and computing $\min_{k\in H_1',l\in H'}, C_{ijkl}$ $\min_{k\in H',l\in H_1'} C_{ijkl}$ or requires $O(tp)$ steps.

Finally, if $k_0 \in H_1$ or $l_0 \in H_1$ then the computation of $T_s'(i, j)$ needs $O(p^2)$ steps. So, the computation of most of the $T_s'(i, j)$ requires $O(p)$ steps in the case where a single hub is exchanged (1-exchange() used in our implementation). Finally, it should be noted that some properties presented in Boland *et al.* (2004) may be used to improve these computations with little impact if all the $T_s'(i, j)$ are computed in parallel.

The overall structure of our system is shown in Fig. 3.

GPU Implementation of the GA

Before presenting the GA, we briefly recall the main features of the GPU and CUDA.

CUDA and GPU

Graphics Processing Units (GPUs) are actually available in most personal computers. They are used to accelerate the execution of a variety of problems. CUDA is a scalable parallel programming model that runs on Nvidias GPU architecture making it possible to use the GPU as a massively parallel machine. Since GPU and CUDA are actually available in most personal computers, massively parallel computing has become a commodity technology. The smallest unit in the GPU that can execute a kernel is called a thread. All threads execute the same code (kernel) but on multiple data. Threads are grouped into blocks and blocks are grouped in the grid. CUDA enables defining grids and blocks according to the parameters of the problem.

Threads can access data in parallel from different memory locations. GPU memory is divided into three levels: (i) Global memory, which can be accessed by all threads. (ii) Shared memory, accessible to all threads of the same block, and (iii) Local memory (register), accessible by a thread and only by it. Shared memory has low latency (2 cycles) and is limited in size. Global memory has high latency (400 cycles) and is large. Each block is divided into Warps of 32 threads that run in parallel on a single Stream Multiprocessor. Programmers must control block size, number of warps, and various memory accesses. Typical CUDA programs are C programs whose functions are classified according to whether they are designed to run on the CPU or on the GPU.

Outline of the GA

GAs is preferred to be executed on parallel architectures such as the GPU since they exploit the availability of many threads performing the same code on multiple data. In the classical thread per chromosome model, one thread performs fitness evaluation as well as genetic operations (Benaini *et al*., 2019). Here, we use several threads to perform these operations on a single chromosome, resulting in better utilization of GPU resources. Our GA uses R blocks, each of $n \times 2n$ threads, to process a population of $2 \times R$ chromosomes as follows.

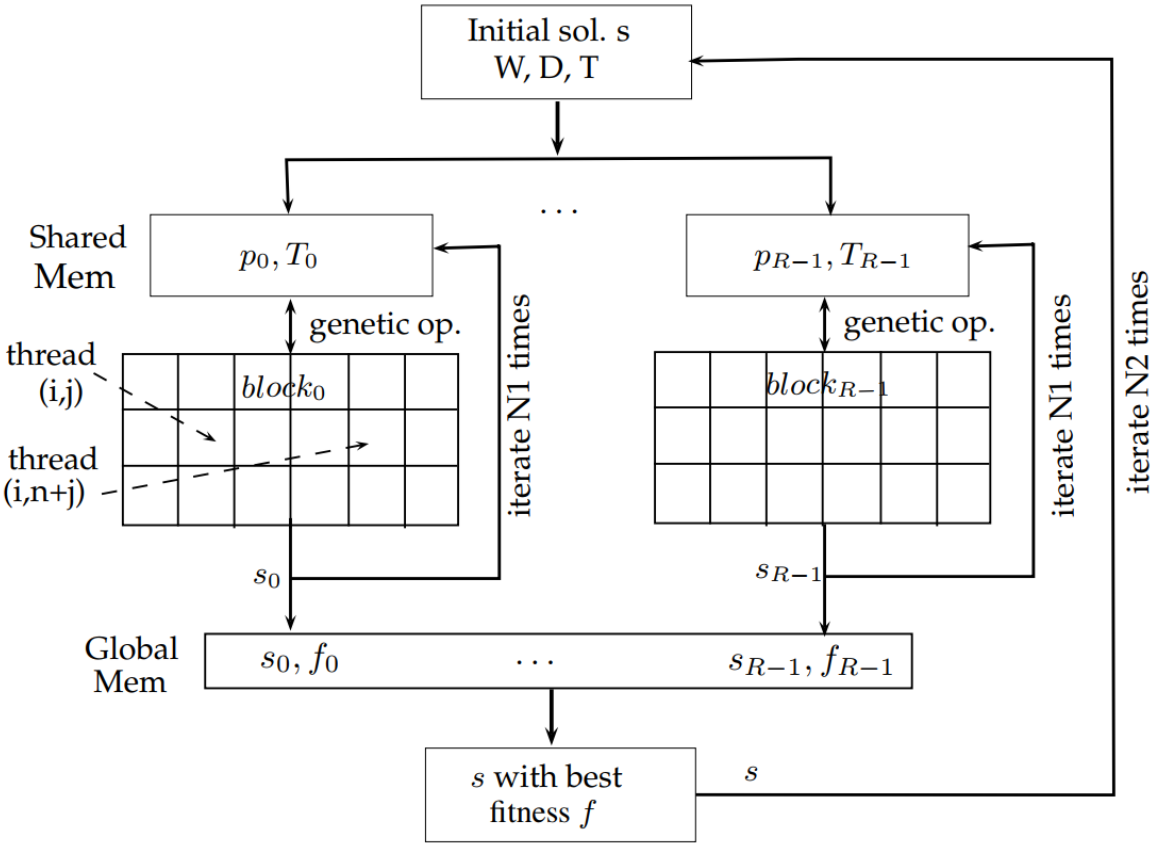

Fig. 4: The parallel GA for the UMApHMP

Each pair of individuals (parents) is processed by a single block. The representations T and T' of the two parents, the W costs matrix, and the D distances matrix are stored in the shared memory of the block (to allow all threads of the block to access these data without going through the global memory). The sub-block composed of threads (i, j) (resp. threads $(i, j + n)$), $0 \leq i, j < n$, updates and evaluates the fitness of one new (resp. the other new) produced individual at each iteration. The fitness of an individual represented by T is simply computed as the $\sum_{i,j} C_{ijT(i,j)}$.

Now, the authors explain the GPU implementation of the GA. Starting from an initial solution s constructed as described. In each block$_i$, $0 \leq i < R$, sets a first parent p_i to s and generates a second parent p'_i = 1-exchange(s). This constitutes a population of $R \times 2$ chromosomes distributed per pair of two per block. Next, each block$_i$ applies the crossover operator to p_i, p'_i producing two feasible children c_i and c'_i, then evaluates their fitness. The individual s_i with the best fitness among p_i, p_i', c_i, c_i' is re-injected in the block$_i$ as the first parent for the next iteration if any. After N_1 iterations (the same for all blocks) the s_i, $0 \leq i < R$, are copied in the global memory. A mutation operator is eventually applied to them and the individual s with the best fitness among them is selected and re-injected in each block as an initial solution for the next iteration if any. The process terminates after a certain number *of* N_2 of iterations. This process is illustrated in Fig. 4 where p_i, f_i, and T_i denote respectively the first parent, its fitness, and its representation.

Finally, note that:

- This implementation has two levels of parallelism. First, in the processing of each pair of parents (p_i, p_i') and secondly in the treatment of the population (p_i, p_i'), $0 \leq i < R$
- The initial solution s is simply generated in parallel by computing the list L and sorting it on a block of $n \times n$ threads. Then determine the p clusters from which the initial hubs are located in $O(n)$ parallel steps
- Other powerful genetic operators proposed especially in Kratica *et al*. (2005); Naeem and Ombuki-Berman (2010) can be implemented in this GA without causing data exchange between the blocks and therefore without additional access to the global memory
- If the shared memory is not enough to store the representations then either one stores them in the global memory or one re-computes all the shortest paths at each iteration or one uses the cache memory technique as in Kratica *et al*. (2005) all these possibilities induce an additional execution time

Results and Discussion

The objective of this section is to give new best results or results for instances that have not been solved before (to our knowledge). Moreover, through these tests, we highlight the relevance of our clustering algorithm et GPU implementation. So, we tested our implementation on the following benchmarks:

- The Australian Post (AP) data set represents mail flows in Australia where the flows are not symmetric, there are flows between each node and itself, and $\chi = 3$, $\gamma = 2$, and $\alpha = 0.75$
- The Urand dataset is random instances of up to 400 nodes generated by Meyer *et al*. (2009), and instances of up to 1000 nodes proposed by Ilić *et al*. (2010) where the node coordinates and the flow matrix were randomly generated. We completed random instances of large sizes up to 6000 that we generated according to the same rules that those of URAND instances
- The planet lab instances are node-to-node delay data for performing measurements of the legacy internet (Ilić *et al*. (2010). In these instances, $\chi = \gamma = \alpha = 1$, the distance matrix does not respect the triangle inequality, and the flows $w_{ij} = 0$ if $i = j$ and 1 otherwise

We used a modest Nvidia quadro to realize our tests. We report our results for the single and for multiple allocation cases for each tested instance. Thus, the costs of the solutions can be compared in the two cases. We compare our single allocation results to those of (Ilić *et al*., 2010) and the multiple allocation results to those of Kratica *et al*., 2005).

The following notations are used in all tables. n is the number of nodes in the instance, and p is the number of hubs. M stands for multiple allocations, and S stands for single allocation. Best sol is the best solution if known, otherwise, the dash is written. GPUsol is our solution with OPT when the solution is known optimal or best the best solution found in the literature. Clust opt is the number of clusters each containing at least one hub of the optimal or the best solution for the UMApHMP. The average of these numbers is given in bold at the end of this column. Some results for the single allocation case (USApHMP) reported here are presented in Benaini *et al*. (2019). For all benchmarks considered our implementation reaches the optimal or the best solutions in execution times $\leq 7s$ for $n \leq 419$ and between 10 and 15s for $n = 1000$ and between 25s and 600s for $n \in [1500,6000]$.

Table 2 provides the results of the AP instances with 300 and 400 nodes. We do not know other UMApHMP solutions for these instances to compare them with our results. The average of the CLUST OPT column is 87%. This means that, on average, the percentage of clusters each containing at least one optimal hub is 87% (or the optimal hubs of each instance problem are distributed on 87% of clusters). This shows the efficiency of our clustering algorithm.

In Table 3, we reported new best costs for Planet Lab instances for the single and multiple allocation cases. The old best solutions for USApHMP reported in this table are due to Ilić *et al*. (2010). Given that there are no other known UMApHMP results for these instances, our results become henceforth the best. The average of the CLUST OPT column is 69%. This percentage is quite low compared to the others benchmarks. We think that this is due to the non-respect of the triangle inequality in the planet lab instances. We complete this table with the results for other values of p in Table 4. Only the results for instances 06, 07, 08, and 09 are reported (to avoid overloading the paper with tables). Note that there are no other known UMApHMP solutions for these instances with these values of p. Here, the percentage of clusters each containing at least one hub of the best solution is quite high.

Ilić *et al*. (2010); Kratica (2013) reported the best-known results for URAND instances for n = 100, 200, 300, 400, and (χ = 3 and 1, α = 0.75, γ = 2 and 1). Our GPU implementation achieves the same results. So, in Table 5 we report only the CLUST to OPT column for these instances with (χ = 1; α = 0.75, γ = 1). We observe that the numbers in this column are not very sensitive to the variation of n. Here too, the average of the CLUST OPT column is quite high. In Table 6 we report new results (for single and multiple allocation cases) on the Urand instances with 1000 nodes with (χ = 1, α = 0.75, γ = 1) since Ilić *et al*. (2010); Kratica (2013) reported only the results for (χ = 3, α = 0.75, γ = 2).

Table 7 reports our results with the parameters (χ = 3, α = 0.75, γ = 1) for the large Urand instances up to 6000 nodes. To our knowledge, no results for instances of sizes larger than 1000 have been published. The efficiency of our clustering algorithm and GPU implementation is clearly established in solving these large random instances. Compared to this study, no article dealing with hub location problems has presented results for instances as large.

Table 2: Results on large AP instances

n	*p*	Type alloc	Best sol	GPU sol	Hub OPT
300	5	M	-	170679.01	5
	10	M	-	131788.68	9
	15	M	-	112806.91	13
	20	M	-	101884.17	17
400	5	M	-	172818.43	5
	10	M	-	133487.54	8
	15	M	-	114314.49	13
	20	M	-	103404.57	17
					87%

Table 3: Results on planet lab instances *-2005

Inst.	*n*	*p*	Type alloc	Best sol	GPU sol	Hub OPT
1	127	12	M	-	2566990	9
			S	2927946	2904434	
2	321	19	M	-	16518458	13
			S	18579238	18329984	
3	324	18	M	-	18238014	12
			S	20569390	20284132	
4	70	9	M	-	682596	6
			S	739954	730810	
5	374	20	M	-	20653586	14
			S	25696352	25583240	
6	365	20	M	-	19365696	14
			S	22214156	22151862	
7	380	20	M	-	27417830	15
			S	30984986	30782956	
8	402	21	M	-	28540846	14
			S	30878576	30636170	
9	419	21	M	-	26593496	13
			S	32959078	32649752	
10	414	21	M	-	26355946	14
			S	32836162	28211380	
11	407	21	M	-	24664598	15
			S	27787880	27644374	
12	414	21	M	-	22317134	14
			S	28462348	28213748	
						69%

Table 4: Results on Planet Lab instances *-2005 for other values of *p*

Inst.	*n*	*p*	Type alloc	GPU sol	Hub OPT	Inst.	*n*	*p*	Type alloc	GPU sol	Hub OPT
6	365	2	M	25569616	2	7	380	2	M	37733504	2
			S	26192564					S	38605660	
		3	M	23799100	3			3	M	34195694	3
			S	25416392					S	26882642	
		4	M	22574576	3			4	M	32060818	4
			S	24528130					S	33874708	
		5	M	21787425	4			5	M	30676098	4
			S	23977334					S	33416162	
		10	M	20630906	8			10	M	28554634	7
			S	22708070					S	31556270	
		15	M	19610164	12			15	M	27813764	11
			S	22398924					S	30958920	
		20	M	19365696	17			20	M	27417830	15
			S	22190348					S	33272248	
8	402	2	M	36642406	2	9	419	2	M	40178280	2
			S	37259814					S	40995336	
		3	M	33907736	3			3	M	36736314	3
			S	35895074					S	39179832	
		4	M	32545826	3			4	M	35088016	4
			S	34535748					S	38151236	
		5	M	31788464	4			5	M	34019204	5
			S	33569834					S	37019968	
		10	M	29699488	8			10	M	31467516	7
			S	32108626					S	34335578	
		15	M	28980246	11			15	M	27208466	13
			S	31008768					S	33178544	
		20	M	28614800	16			20	M	26663202	17
			S	31558084					S	32601346	
											81.7%

Table 5: Results on large Urand instances ($\chi = 1$, $\alpha = 0.75$, $\gamma = 1$)

n	*p*	Clust OPT	*n*	*p*	Clust OPT	*n*	*p*	Clust OPT	*n*	*p*	Clust OPT
100	2	2	200	2	2	300	2	2	400	2	2
	3	3		3	2		3	3		3	3
	4	4		4	3		4	4		5	4
	5	5		5	5		5	4		5	4
	10	9		10	9		10	9		10	8
	15	13		15	12		15	14		15	12
	20	17		20	16		20	17		20	16
											85.6%

Table 6: Results on large URAND instances

n	*p*	Type alloc	GPU sol	Clust
1000	2	M	3380404.48	2
		S	3644705.83	
	3	M	3086100.81	3
		S	3397386.83	
	4	M	2907515.00	4
		S	3206255.10	
	5	M	2825585.52	4
		S	3119060.41	
	10	M	2572595.51	9
		S	2823592.72	
	15	M	2460725.82	14
		S	2679974.67	
	20	M	2393602.47	16
		S	2577234.14	
				88.1%

Table 7: Results on large urand instances generated by us

n	*p*	Type alloc	GPU sol	Clust OPT	*n*	*p*	Type alloc	GPU sol	Clust OPT
1500	20	M	448739772	17	4000	20	M	3163446506	17
		S	454787506				S	3234999192	
	30	M	406017592	26		30	M	2935022500	23
		S	407155164				S	2983891783	
	40	M	377761261	36		40	M	2727794413	33
		S	380114045				S	2769550514	
	50	M	358245622	44		50	M	2604040619	41
		S	363586538				S	2644606684	
2000	20	M	800331930	17	5000	20	M	5001798606	16
		S	805749722				S	5085803132	
	30	M	724383243	27		30	M	4610179267	25
		S	733375448				S	4656787498	
	40	M	679709297	37		40	M	4277839570	32
		S	686515363				S	4353561395	
	50	M	647033467	44		50	M	4060895916	39
		S	655938000				S	4143849388	
3000	20	M	1777794117	17	6000	20	M	7217166936	15
		S	1804950952				S	7398401957	
	30	M	1616819166	24		30	M	6586643628	24
		S	1642145354				S	6675723961	
	40	M	1522572071	33		40	M	6197499299	34
		S	1538548764				S	6293053841	
	50	M	1445702469	42		50	M	5902887457	43
		S	1468780124				S	5999780197	
									84%

Conclusion

The authors proposed a GPU-based approach for solving the multiple allocations p-hub median problems with a parallel genetic algorithm. The initial solutions of the GA are generated using a simple and efficient clustering algorithm that partitions the set of nodes in *p* clusters, each likely to contain an optimal hub. This clustering algorithm seems very efficient at least for the tested benchmarks. However, we failed to define an efficient criterion, common to all these benchmarks, to identify the potential hub in each cluster. The initial solutions thus generated are improved by the genetic algorithm. A binary encoding and two-dimensional array that specifies all pair shortest paths between origin-destination nodes are used by the GA. The most expensive operation of the GA is the updating of the shortest cost paths (representation *T*) after each change of the hubs. This was efficiently done by a block of threads and without re-computing all pair shortest paths.

Exploiting the enormous computational power of the GPU, our CUDA program obtains solutions that match with optimal ones known and outperforms the previously published results for the UMApHMP. Moreover, it solved instances of problems so far unsolved (URAND instances of size up to 6000 nodes). Obviously, this program executed on a more powerful GPU such as the Nvidia-Kepler will give better results in time and quality and especially, it will allow to solve large hard instances which require the exhaustion of the entire search space AlBdaiwi and AboElFotoh (2017). Future studies might consider studying other versions of the hub location problem including robust and dynamic problems. Improving the clustering process and defining an efficient criterion for selecting the optimal hub in each cluster is another future work.

Acknowledgment

The authors would like to thank the Editor-in-chief and anonymous reviewers for their comments and suggestions to improve the quality of the paper.

Funding Information

The authors have not received any financial support or funding to report.

Author's Contributions

Achraf Berrajaa: Collect data and benchmarks, implement the proposed algorithm. Written of the article with emphasis on the concepts, the proposed algorithm, the limits of the proposed approaches. Drafted of the manuscript and designed of the figures. Explore research issues and challenges.

Ayyoub El Outmani: Reviewed the final version of the manuscript. Provided valuable feedback to enhance the quality of the study.

Abdelhamid Benaini: Conceptualize the work,

supervised the construction and wrote part of the paper.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

AlBdaiwi, B. F., & AboElFotoh, H. M. (2017). A GPU-based genetic algorithm for the p-median problem. *The Journal of Supercomputing*, *73*, 4221-4244. https://doi.org/10.1007/s11227-017-2006-x

Alvarez Fernandez, S., Ferone, D., Juan, A., & Tarchi, D. (2022). A simheuristic algorithm for video streaming flows optimisation with QoS threshold modelled as a stochastic single-allocation p-hub median problem. *Journal of Simulation*, *16*(5), 480-493. https://doi.org/10.1080/17477778.2020.1863754

Benaini, A., & Berrajaa, A. (2020, October). Parallel genetic algorithm on gpu for the robust uncapacitated single allocation p-hub median problem with discrete scenarios. In *2020 5th International Conference on Logistics Operations Management (GOL)* (pp. 1-8). IEEE. https://doi.org/10.1109/GOL49479.2020.9314716

Benaini, A., Berrajaa, A., & Boukachour, J. (2022, June). GPU computing for the capacitated single allocation hub location problem. In *2022 IEEE 6th International Conference on Logistics Operations Management (GOL)* (pp. 1-8). IEEE. https://doi.org/10.1109/GOL53975.2022.9820511

Benaini, A., Berrajaa, A., Boukachour, J., & Oudani, M. (2019). Solving the uncapacitated single allocation p-hub median problem on gpu. *Bioinspired Heuristics for Optimization*, 27-42. https://doi.org/10.1007/978-3-319-95104-1_2

Boland, N., Krishnamoorthy, M., Ernst, A. T., & Ebery, J. (2004). Preprocessing and cutting for multiple allocation hub location problems. *European Journal of Operational Research*, *155*(3), 638-653. https://doi.org/10.1016/S0377-2217(03)00072-9

Chen, J. F. (2006). A heuristic for the uncapacitated multiple allocation hub location problem. *Journal of the Chinese Institute of Industrial Engineers*, *23*(5), 371-381. https://doi.org/10.1080/10170660609509333

Contreras, I., & O'Kelly, M. (2019). Hub location problems. *Location Science*, 327-363. https://doi.org/10.1007/978-3-030-32177-2_12

Demir, İ., Kiraz, B., & Ergin, F. C. (2022). Experimental evaluation of meta-heuristics for multi-objective capacitated multiple allocation hub location problem. *Engineering Science and Technology, an International Journal*, *29*, 101032. https://doi.org/10.1016/j.jestch.2021.06.012

Ilić, A., Urošević, D., Brimberg, J., & Mladenović, N. (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. https://doi.org/10.1016/j.ejor.2010.02.022

Kratica, J., Stanimirović, Z., Tošić, D., & Filipović, V. (2005). Genetic algorithm for solving uncapacitated multiple allocation hub location problem. *Computing and Informatics*, *24*(4), 415-426.

Kratica, J. (2013). An electromagnetism-like metaheuristic for the uncapacitated multiple allocation p-hub median problem. *Computers & Industrial Engineering*, *66*(4), 1015-1024. https://doi.org/10.1016/j.cie.2013.08.014

Lim, G. J., & Ma, L. (2013). GPU-based parallel vertex substitution algorithm for the p-median problem. *Computers & Industrial Engineering*, *64*(1), 381-388. https://doi.org/10.1016/j.cie.2012.10.008

Rabbani, M., & Kazemi, S. (2015). Solving uncapacitated multiple allocation p-hub center problem by Dijkstra's algorithm-based genetic algorithm and simulated annealing. *International Journal of Industrial Engineering Computations*, *6*(3), 405-418. https://doi.org/10.5267/j.ijiec.2015.2.002

Meyer, T., Ernst, A. T., & Krishnamoorthy, M. (2009). A 2-phase algorithm for solving the single allocation p-hub center problem. *Computers & Operations Research*, *36*(12), 3143-3151. https://doi.org/10.1016/j.cor.2008.07.011

Naeem, M., & Ombuki-Berman, B. (2010, July). An efficient genetic algorithm for the uncapacitated single allocation hub location problem. In *IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE. https://doi.org/10.1109/CEC.2010.5586382

Peker, M., Kara, B. Y., Campbell, J. F., & Alumur, S. A. (2016). Spatial analysis of single allocation hub location problems. *Networks and Spatial Economics*, *16*, 1075-1101. https://doi.org/10.1007/s11067-015-9311-9

Rouzpeykar, Y., Soltani, R., & Afashr Kazemi, M. A. (2022). EFP-GA: An Extended Fuzzy Programming Model and a Genetic Algorithm for Management of the Integrated Hub Location and Revenue Model under Uncertainty. *Complexity*, *2022*. https://doi.org/10.1155/2022/7801188

Sangsawang, O. (2011). Metaheuristics for hub location models. https://tigerprints.clemson.edu/all_dissertations/804

Santos, L., Madeira, D., Clua, E., Martins, S., & Plastino, A. (2010). A parallel GRASP resolution for a GPU architecture. In *International Conference on Metaheuristics and Nature Inspired Computing, META10*. https://www.researchgate.net/publication/2792633

Sender, J., & Clausen, U. (2013). Heuristics for solving a capacitated multiple allocation hub location problem with application in German wagonload traffic. *Electronic Notes in Discrete Mathematics*, *41*, 13-20. https://doi.org/10.1016/j.endm.2013.05.070

Shobeiri, A. (2015). *Grasp metaheuristic for multiple allocation p-hub location problem* (Doctoral dissertation, Concordia University). https://spectrum.library.concordia.ca/id/eprint/981261/

Wang, Y., & Huang, Y. (2016). Incremental optimization of hub and spok network under changes of spokes number and flow. *Int. J. Hybrid information Technology*, *9*(1), 339352. https://doi.org/10.14257/ijhit.2016.9.1.29

Yaman, H. (2011). Allocation strategies in hub networks. *European Journal of Operational Research*, *211*(3), 442-451. https://doi.org/10.1016/j.ejor.2011.01.014

Zetina, C. A., Contreras, I., Cordeau, J. F., & Nikbakhsh, E. (2017). Robust uncapacitated hub location. *Transportation Research Part B: Methodological*, *106*, 393-410. https://doi.org/10.1016/j.trb.2017.06.008

Zhang, C., Sun, X., Dai, W., & Wandelt, S. (2023). Solving Hub Location Problems with Profits Using Variable Neighborhood Search. *Transportation Research Record*, *2677*(1), 1675-1695. https://doi.org/10.1177/03611981221105501